

A4

1/5/1
DIALOG(R) File 351:Derwent WPI
(c) 2000 Derwent Info Ltd. All rts. reserv.

011744611 **Image available**
WPI Acc No: 1998-161521/199815
XRPX Acc No: N98-128498

Data processing apparatus with data processing being performed in selected node of network - has control device for migrating executable process from node to another node in response to certain result of execution by executable process

Patent Assignee: TOSHIBA KK (TOKE)

Inventor: HATTORI M; HONIDEN S; IRIE Y; KAGAYA A; NAGAI Y; OHSUGA A; TAHARA Y

Number of Countries: 019 Number of Patents: 002

Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week
EP 829806	A2	19980318	EP 97116164	A	19970917	199815 B
JP 10149287	A	19980602	JP 97176181	A	19970701	199832

Priority Applications (No Type Date): JP 97176181 A 19970701; JP 96244688 A 19960917

Cited Patents: No-SR.Pub

Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
-----------	------	-----	----	----------	--------------

EP 829806	A2	E	67	G06F-009/46	
-----------	----	---	----	-------------	--

Designated States (Regional): AT BE CH DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE

JP 10149287	A		48	G06F-009/44	
-------------	---	--	----	-------------	--

Abstract (Basic): EP 829806 A

The apparatus includes a first storage device for storing a first information representing a state of constituent elements of the node to be accessed by the executable process. A second storage device is used for storing a second information representing actions of the executable process. A plan- generating device generates a plan representing a series of actions of the executable process using the first and second information in response to a given request code. An execution device executes the series of actions of the executable process to the constituent elements based on the generated plan.

A control device is used for migrating the executable process from the node to another node in response to a certain result of the execution by the executable process.

USE - In data processing apparatus, which processes information that exist in distributed form on network.

ADVANTAGE - Provides superior immunity to line faults. Generates plan efficiently by connecting actions, based on pre-condition and post-condition for each action. Capable of flexible operation.

Dwg.1/22

Title Terms: DATA; PROCESS; APPARATUS; DATA; PROCESS; PERFORMANCE; SELECT; NODE; NETWORK; CONTROL; DEVICE; MIGRATION; EXECUTE; PROCESS; NODE; NODE; RESPOND; RESULT; EXECUTE; EXECUTE; PROCESS

Derwent Class: T01

International Patent Class (Main): G06F-009/44; G06F-009/46

International Patent Class (Additional): G06F-013/00; G06F-015/16

File Segment: EPI

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平10-149287

(43) 公開日 平成10年(1998) 6月2日

(51) Int.Cl.⁶

G 0 6 F 9/44
13/00
15/16

識別記号

5 5 2
3 5 1
3 5 5
4 3 0

F I

G 0 6 F 9/44
13/00
15/16

5 5 2
3 5 1 H
3 5 5
4 3 0 B

審査請求 未請求 請求項の数15 O L (全 48 頁)

(21) 出願番号 特願平9-176181

(22) 出願日 平成9年(1997) 7月1日

(31) 優先権主張番号 特願平8-244688

(32) 優先日 平8(1996) 9月17日

(33) 優先権主張国 日本 (J P)

(71) 出願人 000003078

株式会社東芝

神奈川県川崎市幸区堀川町72番地

(72) 発明者 田原 康之

神奈川県川崎市幸区柳町70番地 株式会社
東芝柳町工場内

(72) 発明者 大須賀 昭彦

神奈川県川崎市幸区柳町70番地 株式会社
東芝柳町工場内

(72) 発明者 永井 保夫

神奈川県川崎市幸区柳町70番地 株式会社
東芝柳町工場内

(74) 代理人 弁理士 木内 光春

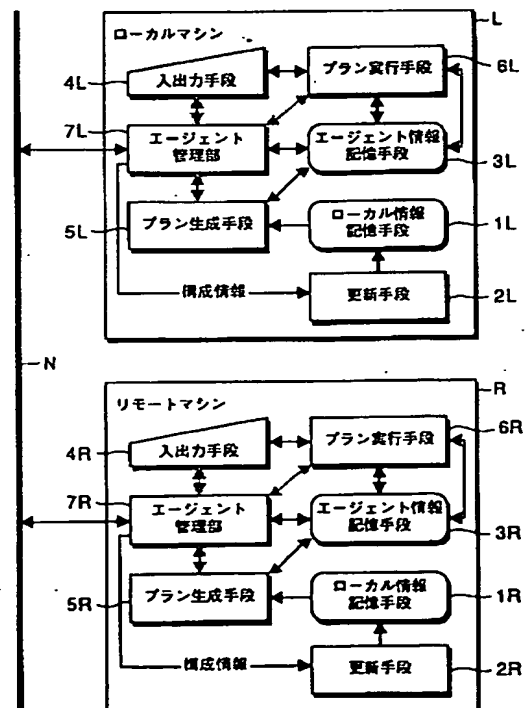
最終頁に続く

(54) 【発明の名称】 情報処理装置、情報処理方法及び記録媒体

(57) 【要約】

【課題】 ソフトウェアの構成要素の変化に柔軟に対応し、かつ、回線障害に対する耐障害性を向上させる。

【解決手段】 各ノードL, Rのローカル情報記憶手段1に、構成要素へのアクセスに用いるローカル情報を格納し、更新手段2によって更新する。プラン生成手段5が、入力された前記要求記述を満たすためにエージェントがとるべき行動を表すプランを、エージェント情報及びローカル情報に基づいて、複数のアクションの集合として生成する。プラン実行手段6が、生成された前記プラン中の各アクションに基づいて各ノードL, R上でエージェントの動作を実現し、エージェント管理部7が、プラン中のgoアクションに基づいて、前記エージェントをノード間で移動させる。移動先のノードでも、プラン実行の失敗時など、必要に応じて再プランニングや子エージェントの生成が行われる。



【特許請求の範囲】

【請求項1】 ソフトウェア上の実行主体がネットワークに接続されたノード上で動作することによって情報処理を行う情報処理装置において、

前記ノードに配置された構成要素へのアクセスに用いるための構成要素の状態を表現する第1の情報を記憶する第1の記憶手段と、

前記実行主体の挙動を定義する第2の情報を記憶するための第2の記憶手段と、

前記情報処理に対して与えられた要求記述を満たすために実行主体がとるべき行動を表すプランを、前記第1の情報及び前記第2の情報に基づいて、アクションの集合として、生成する生成手段と、

生成された前記プランに基づいてノード上で実行主体の動作を実現する実行手段と、

前記生成された前記プランを構成する前記アクションのうちの第1のアクションに基づいて、前記実行主体をノード間で移動させる管理手段と、

を有することを特徴とする情報処理装置。

【請求項2】 前記第2の記憶手段は、

実行主体の動作の単位であるアクションの集合を記憶する処理と、

前記アクションが実行可能となることを表す事前条件と、実行による状態変化の内容を表す事後条件とを含むアクション情報を記憶する手段と、

アクションの実行に関する所定の制約条件を記憶する手段と、

各アクション間の因果関係を表す情報を記憶する手段と、

を備え、

前記生成手段は、前記アクションの集合中の所定のアクションに関し、前記アクション情報に基づいて、新たな制約条件又は前記因果関係を表す情報のうち少なくとも一方を、前記第2の記憶手段に記憶させる手段を備えたことを特徴とする請求項1記載の情報処理装置。

【請求項3】 前記生成手段は、

特定のノードで実行すべきアクションの事前条件を事後条件として実行主体が前記特定のノードへ移動するアクションを前記アクションの集合に追加し、

アクションの集合に含まれる各アクションの事前条件のうち、前記因果関係を表す情報において成立する事実として記録されていないものをサブゴールとして順次選択し、選択したサブゴールの事前条件を達成する事後条件を持つアクションをさらに順次選択し、

選択された各アクションについて前記新たな制約条件又は前記因果関係を表す情報のうち少なくとも一方を、前記第2の記憶手段に記憶させるように構成されたことを特徴とする請求項2記載の情報処理装置。

【請求項4】 未達成ゴールが存在しない場合は、プランの生成が成功で、プランを実行するまでもなく要求記

述が達成されたと判定し、

未達成ゴールが存在しかつアクション集合が空でない場合は、プランの生成は成功でかつプランの実行が必要と判定する判定手段を有することを特徴とする請求項1記載の情報処理装置。

【請求項5】 前記実行主体をノード上のプロセスとして設定し、

各ノードの前記各管理手段は、転送元となるノードから転送先となるノードへ実行主体を転送するときに、

転送元から転送先へ実行主体の受け入れ要求を送信し、受け入れ要求を受信した転送先において、実行主体用のプロセスを設定することによって実行主体を受け入れる準備を行ない、

転送先から転送元に準備完了の通知を送信し、

準備完了の通知を受信した転送元から前記第1の情報を転送先に送信し、

第1の情報を受信した転送先で、受信した第1の情報に基づいて前記プロセスの実行を開始し、

転送元において実行主体のプロセスを終了するように構成されたことを特徴とする請求項1記載の情報処理装置。

【請求項6】 ソフトウェア上の実行主体がネットワークに接続されたノード上で動作することによって情報処理を行う情報処理方法において、

前記ノードに配置された構成要素へのアクセスに用いるための構成要素の状態を表現する第1の情報を記憶する第1の記憶処理と、

前記記憶されている所定の情報を更新する更新処理と、

前記実行主体の挙動を定義する第2の情報を記憶するための第2の記憶処理と、

前記情報処理に対する要求記述を入力するための入力処理と、

前記第1の情報及び前記第2の情報に基づいて、入力された前記要求記述を満たすために実行主体がとるべき行動を表すプランを、アクションの集合として生成する生成処理と、

生成された前記プランに基づいてノード上で実行主体の動作を実現する実行処理と、

前記生成された前記プランを構成する前記アクションのうちの第1のアクションに基づいて、前記実行主体をノード間で移動させる管理処理と、

を含むことを特徴とする情報処理方法。

【請求項7】 ソフトウェア上の実行主体がネットワークに接続されたノード上で動作することによって情報処理を行う際に、

前記ノードに配置された構成要素へのアクセスに用いるための構成要素の状態を表現する第1の情報を記憶する第1の記憶処理と、

前記記憶されている所定の情報を更新する更新処理と、前記実行主体の挙動を定義する第2の情報を記憶するた

めの第2の記憶処理と、

前記情報処理に対する要求記述を入力するための入力処理と、

前記第1の情報及び前記第2の情報に基づいて、入力された前記要求記述を満たすために実行主体がとるべき行動を表すプランを、アクションの集合として生成する生成処理と、

生成された前記プランに基づいてノード上で実行主体の動作を実現する実行処理と、

前記生成された前記プランを構成する前記アクションのうちの第1のアクションに基づいて、前記実行主体をノード間で移動させる管理処理と、

を含む情報処理方法を実現するコンピュータプログラムを記録したことを特徴とする記録媒体。

【請求項8】 ソフトウェア上の実行主体であるエージェントがネットワークに接続されたノード上で動作することによって情報処理を行う情報処理装置において、前記ノードに配置された構成要素へのアクセスに用いるための構成要素の状態を表現する第1の情報と、前記実行主体の挙動を定義する第2の情報を記憶するための手段と、

前記第1及び第2の情報に基づいて、与えられた目的を達成するためにエージェントが実行すべきプランを作成する手段と、

作成されたプランの実行及びエージェントのノード間における移動を実現するための手段と、を有し、

移動先のノードにおいても、プランの実行及び必要なプランの作成を行うように構成されたことを特徴とする情報処理装置。

【請求項9】 各ノードに配置された前記各情報が、エージェントの種類ごとに対応する複数のフィールドに区分され、

エージェントのプランは、当該エージェントに対応するフィールドの情報に基づいて作成され、

ノード間における移動では、エージェントが移動先のノードにおける対応する前記フィールドに移動するように構成されたことを特徴とする請求項8記載の情報処理装置。

【請求項10】 エージェントが、当該エージェントに固有の状態又はアクションに関する情報を持ち、ノード間の移動の際もこれら情報を運搬し、運搬しているこれら情報を移動した先のノードにおける前記第1の情報及び前記第2の情報と合わせてプランニングに用いるように構成されたことを特徴とする請求項8又は9記載の情報処理装置。

【請求項11】 前記プランを作成する手段は、プランの一部としてエージェントをノード間で移動させる移動命令又はその他の命令を作成する際に、当該命令の前提となる事前条件を当該移動命令又は当該その他の命令に添付するように構成されたことを特徴とする請求項8、

9又は10記載の情報処理装置。

【請求項12】 前記エージェントが、最終的な目的であるゴールを達成するための中間的なサブゴールを作成し、このサブゴールを達成するための子エージェントを生成するように構成されたことを特徴とする請求項8、9、10又は11記載の情報処理装置。

【請求項13】 ソフトウェア上の実行主体であるエージェントがネットワークに接続されたノード上で動作することによって情報処理を行う情報処理方法において、前記ノードに配置された構成要素へのアクセスに用いるための構成要素の状態を表現する第1の情報と、前記実行主体の挙動を定義する第2の情報を記憶するステップと、

前記第1及び第2の情報に基づいて、与えられた目的を達成するためにエージェントが実行すべきプランを作成するステップと、

作成されたプランの実行及びエージェントのノード間における移動を実現するためのステップと、を有し、移動先のノードにおいても、プランの実行及び必要なプランの作成を行うことを特徴とする情報処理方法。

【請求項14】 ソフトウェア上の実行主体であるエージェントがネットワークに接続されたノード上で動作することによって情報処理を行う情報処理方法を実現するコンピュータプログラムを記録した記録媒体において、当該情報処理方法は、

前記ノードに配置された構成要素へのアクセスに用いるための構成要素の状態を表現する第1の情報と、前記実行主体の挙動を定義する第2の情報を記憶するステップと、

前記第1及び第2の情報に基づいて、与えられた目的を達成するためにエージェントが実行すべきプランを作成するステップと、

作成されたプランの実行及びエージェントのノード間における移動を実現するためのステップと、を有し、移動先のノードにおいても、プランの実行及び必要なプランの作成を行うことを特徴とする記録媒体。

【請求項15】 請求項13記載の情報処理方法を実現するコンピュータプログラムを記録した記録媒体において、

当該情報処理方法は、

各ノードに配置された前記各情報が、エージェントの種類ごとに対応する複数のフィールドに区分され、

エージェントのプランは、当該エージェントに対応するフィールドの情報に基づいて作成され、

ノード間における移動では、エージェントが移動先のノードにおける対応する前記フィールドに移動することを特徴とする記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、ネットワーク上に

分散して存在する情報を処理する情報処理装置及び情報処理方法の改良に関するものである。

【0002】

【従来の技術】近年、コンピュータを中心とした情報処理システム技術においては、ダウンサイジングの進行や、インターネットなどネットワーク環境の整備が進んでいる。このため、情報処理の装置や方法における主流は、データファイルや機能ライブラリなどの構成要素がネットワーク上の各マシンに分散した分散システムに移行しつつある。これに伴い、企業や研究所などのローカルネットワークでも、広域ネットワークとの接続によって環境のオープン化がより進展している。このように広域ネットワークと接続されたネットワークを開放型ネットワークと呼ぶ。

【0003】開放型ネットワークに代表される大規模ネットワークでは、しばしば、複数のマシン上に分散したいくつかのデータファイルや機能ライブラリなどの構成要素を組み合わせることによって、一つのソフトウェアを構成する。このように構成されるソフトウェアを分散システムと呼ぶ。分散配置された構成要素は、マシンすなわちノードに関する管理上の事情や、ネットワークに関する管理上の事情などで、別のマシンやディレクトリに移動されたり、名称やアクセス権などの属性が変更されることが考えられる。このような変更は、次のような問題を生ずる。

【0004】まず、ソフトウェアに処理を要求するときは、メッセージやコマンドなどの要求記述を入力し、この入力の際に、あるマシン上に配置されている処理に関わる特定の構成要素が指定される。しかし、指定された構成要素が他のマシンに移動されると、移動先の新しいマシンを指定するために、ソフトウェアに対する入力をやり直すか、修正しなければならない。特に、ソフトウェアの一部が、移動された構成要素にアクセスするように構成されている場合は、構成要素の移動は、ソフトウェアの構成そのものの変化を意味する。この場合、アクセスする側のソフトウェアの部分を変更して、移動先の新しいマシンを指定しなければならない。

【0005】従来では、マシン・機能・ファイルなどは、具体的に名指しされていたため、機能又はファイルが移動されると、変更に合わせてソフトウェアや入力を変更しなければならなかった。しかも、このような指定は処理開始前に行なう必要があった。

【0006】分散システムにおいて、ユーザに対してサービスを確実に提供するには、こうした変化に対して適応する柔軟なソフトウェアを構築することが重要である。特に、このような変化に対応して、処理開始後であっても指定を変更し、かつこの変更は極力人手を介さず自動的に行われることが望ましい。

【0007】ネットワークにおいて、このような柔軟なソフトウェアのアーキテクチャを実現する技術として、

近年、エージェント指向技術が注目され、数多くの試みがなされている。エージェント指向技術は、エージェントを単位としてソフトウェアを構成しようとするソフトウェア開発技術であり、ここでいうエージェントは、ソフトウェア上の処理の単位で、環境の変化に自律的かつ柔軟に対応するものである。

【0008】たとえば特開平7-182174では、リモートプログラミングの実施方法を開示している。リモートプログラミングは、分散システムにおいて、処理が開始されたノード（ローカルマシンと呼ぶ）から、エージェントが他のノード（リモートマシンと呼ぶ）に転送されるようなプログラミングである。

【0009】エージェントが活動するには、インストラクションと内部情報が必要である。インストラクションは、エージェントの動作（振る舞い）を記述したもので、内部情報は、エージェントの動作によって操作される情報である。内部情報は、より具体的には、変数や配列のほかスタック、バッファ、キュー、フラグなど、エージェントの動作に関連する一切の情報を含む。インストラクション及び内部情報を合わせてエージェント情報と呼ぶ。

【0010】インストラクションは、ファイルのオープンやクローズなど、さまざまな動作の単位ごとに記述される。リモートプログラミングでは、特殊なインストラクションとして、goオペレーションが用いられる。goオペレーションは、エージェントをリモートマシンに転送する処理を行なわせるもので、例えば、ある処理中に、別のマシンにあるファイルに対するインストラクションが含まれる場合は、そのインストラクションに先立って、goオペレーションが記述され、実行される必要がある。

【0011】このようなエージェントを用いるリモートプログラミングを実施する装置の一例について、構成を表す機能ブロック図を図23に示す。この装置では、ネットワークNに接続されたローカルマシンLとリモートマシンRが相互に同様の構成を有し、プロセスとしてエージェントを取り扱う。なお、ここでいうプロセスは、オペレーティングシステムによって管理の対象となる処理の単位であり、複数のプロセスを同時に管理することをマルチプロセスと呼ぶ。

【0012】エージェント管理手段51L、51Rは、このようなエージェントについて、資源管理、設定、終了、スケジューリング及び転送など、エージェント自身を対象とする処理を司る。エージェントによる処理を開始するには、まず、エージェント情報を、ローカルマシンL上のエージェント情報記憶手段31Lに格納する。エージェント情報をエージェント情報記憶手段31Lに格納するには、入出力手段41Lから入力したり、図示しない外部記憶装置からロードするなどすればよい。

【0013】また、入出力手段41Lからエージェント

による処理の開始が指示されると、解釈実行手段61Lが、エージェント情報記憶手段31L内のインストラクションを逐次解釈実行することによって処理を進め、エージェント情報記憶手段31L内の内部情報を操作する。インストラクション中のgoオペレーションが解釈実行されると、解釈実行手段61Lがその旨をエージェント管理手段51Lに通知し、エージェント管理手段51Lは次のようなエージェントを転送する処理を行う。

【0014】まず、エージェント管理手段51Lは、ネットワークNを通じてリモートマシンR上のエージェント管理手段51Rにエージェントの受け入れ準備を要求する。要求を受けたエージェント管理手段51Rは、資源の割当や、エージェントとして管理するプロセスの設定など、エージェントの受け入れ準備を行なった後、ローカルマシンL上のエージェント管理手段51Lに準備完了を通知する。

【0015】この通知を受けたエージェント管理手段51Lは、エージェント情報記憶手段31L内のインストラクションと、上記goオペレーションの解釈実行時におけるエージェントの内部状態を読み出し、双方をリモートマシンRに転送する。リモートマシンR上のエージェント管理手段51Rは、転送されたインストラクションと内部状態をエージェント情報記憶手段31Rに格納したうえ、解釈実行手段61Rにその旨を通知し、解釈実行を開始させる。

【0016】エージェントの転送が無事に終了すると、ローカルマシンL側のエージェント管理手段51Lはプロセスを終了し、不要になったメモリやCPU時間など資源を開放する。

【0017】転送された側のリモートマシンRでは、エージェント情報記憶手段31R内のインストラクションと内部状態を用いて、処理が続行される。なお、続行された処理がリモートマシンR上で終了する場合もあれば、エージェントがもとのローカルマシンLに再度転送される場合もありうる。エージェントが再度転送されると、インストラクションの実行は再度ローカルマシンL上で行われることになる。以上のようなリモートプログラミングによって、分散システム上における柔軟な処理が実現される。

【0018】また、このようなリモートプログラミングにおいて、各マシン間で共通のインストラクション及び内部情報を用いながら、解釈実行手段やエージェント管理手段などを各マシン固有のハードウェアやOSに合わせて構成すれば、構成の異なるマシン間で互換性が実現される。このような構成により、オペレーティングシステムおよびハードウェアが異なるコンピュータシステムにおいて、インストラクションを移動し実行できるなど、上述のような柔軟な対応が可能となる。

【0019】一方、ネットワーク上の複数のノードにまたがって処理を行なうための別の従来技術として、次の

ようなものも知られている（参考文献：Oren Etzioni and Daniel Weld "A Softbot-Based Interface to the Internet" (Communications of ACM)）。この技術では、ファイルを転送を行なうftp、遠隔ログインのための仮想端末機能を実現するtelnetなど、通常のネットワーク機能を利用して、複数のマシンにまたがった処理を行なう。そして、動作中に収集した情報に基づき、ソフトウェアによって自動的に各機能を試行錯誤的に利用し、状況に応じて柔軟にプランニングを行なうことによって、ファイルなどの構成変化に対応した機能選択を行う。

【0020】例えば、目的の機能が他のノードに移転された場合、移転前のノードでその機能にアクセスしようとする失敗するので、元のマシン（ノード）においてプランニングを行ない、アクセス先を第2候補に変更し再度アクセスする。この技術によれば、システムの各時点での状態に対応して、柔軟な動作が可能である。なお、telnetやftpなどの利用は互換性が予め判明している範囲内で行なう。

【0021】

【発明が解決しようとする課題】しかしながら、上記のような従来技術には次のような問題点が存在していた。まず、リモートプログラミングの実施方法（特開平7-182174）では、エージェントの動作系列をインストラクションとして全て利用者が記述しなければならなかったため、プログラミング作業が煩雑である上、エージェントの柔軟性に限界があった。また、goオペレーションによるエージェントの移動先や、利用すべき構成要素が存在するマシンのノード名（ネットワークにおける識別子）のような、ソフトウェアの構成要素の変化に対応するためには、このような変化を捕捉する必要があるが、従来はこのような変化に対応する手段は備えられていなかった。さらに、構成要素の変化に対応するには、変化に応じてインストラクションを変更する必要があるが、従来は、アクセス先のマシン名が具体的に名指しされていたため、アクセス先を変更しなければ目的を達成できない場合においても、その場でインストラクションを変更することは不可能であった。このため、構成要素の変化に柔軟に対応する技術が求められていた。特に、開放型ネットワークのような大規模ネットワークになるほど変化が頻繁になるため、変化への対応が強く求められていた。

【0022】また、Etzioniらの方法では、処理中に、ローカルマシンとリモートマシンとの間で相互にアクセスを続け、継続して情報をやり取りする必要がある。このため、処理の途中で回線障害が生じ、アクセスのルートが切断されると正常な処理の続行ができない、という問題があった。また、遠隔ノードにおける情報の内容をきめこまかく参照して処理内容を変えたり、遠隔ノードの情報へのアクセスを何度も実施する必要がある

場合、また、処理に何らかのリアルタイム性が要求される場合などは、リモートマシン上のプロセスとして情報処理を行ったほうが効率的な場合もある。

【0023】本発明は、上記のような従来技術の問題点を解決するために提案されたもので、その目的は、ソフトウェアの構成要素の変化に柔軟に対応し、かつ、回線障害に対する耐障害性に優れた情報処理装置及び情報処理方法を提供することである。より具体的には、本発明の目的は、ソフトウェアの構成要素に関する情報に基づいて柔軟に作成したプランによって、実行主体たるエージェントの行動を定めることである（請求項1、6、7）。また、本発明の他の目的は、アクションごとの事前条件と事後条件に基づいて、アクションを接続していくことによって、効率的にプランを生成することである（請求項2、3）。また、本発明の他の目的は、エージェントをプロセスとして実現することによって、並行処理とシステム資源の有効活用を図ることである（請求項5）。

【0024】また、本発明の他の目的は、柔軟な運用が可能な情報処理装置及び情報処理方法を提供することである。また、本発明の他の目的は、処理の効率に優れた情報処理装置及び情報処理方法を提供することである。より具体的には、本発明の目的は、移動先のノードでもプラン作成を行うことによって、柔軟な情報処理を実現することである（請求項8、13、14）。また、本発明の他の目的は、プラン作成に用いる情報を複数のフィールドに区分しておき、エージェントの種類に対応するフィールドの情報のみを用いてプラン作成を行うことによって、プラン作成を効率化することである（請求項9、15）。また、本発明の他の目的は、移動命令に事前条件を添付しておくことによって、移動の失敗への対応を容易にすることである（請求項11）。また、本発明の他の目的は、プランの実行や移動が失敗しても、再度プランを作成することによって、情報処理を円滑に継続させることである。

【0025】

【課題を解決するための手段】上記の目的を達成するため、本発明は、次のような構成及び作用を有する。まず、請求項1の発明は、ソフトウェア上の実行主体がネットワークに接続されたノード上で動作することによって情報処理を行う情報処理装置において、前記ノードに配置された構成要素へのアクセスに用いるための構成要素の状態を表現する第1の情報を記憶する第1の記憶手段と、前記記憶されている所定の情報を更新する更新手段と、前記実行主体の挙動を定義する第2の情報を記憶するための第2の記憶手段と、前記情報処理に対して与えられた要求記述を満たすために実行主体がとるべき行動を表すプランを、前記第1の情報及び前記第2の情報に基づいて、アクションの集合として、生成する生成手段と、生成された前記プランに基づいてノード上で実行

主体の動作を実現する実行手段と、前記生成された前記プランを構成する前記アクションのうちの第1のアクションに基づいて、前記実行主体をノード間で移動させる管理手段と、を有することを特徴とする。

【0026】請求項6の発明は、請求項1の発明を方法の観点から把握したものであって、ソフトウェア上の実行主体がネットワークに接続されたノード上で動作することによって情報処理を行う情報処理方法において、前記ノードに配置された構成要素へのアクセスに用いるための構成要素の状態を表現する第1の情報を記憶する第1の記憶処理と、前記記憶されている所定の情報を更新する更新処理と、前記実行主体の挙動を定義する第2の情報を記憶するための第2の記憶処理と、前記情報処理に対する要求記述を入力するための入力処理と、前記第1の情報及び前記第2の情報に基づいて、入力された前記要求記述を満たすために実行主体がとるべき行動を表すプランを、アクションの集合として生成する生成処理と、生成された前記プランに基づいてノード上で実行主体の動作を実現する実行処理と、前記生成された前記プランを構成する前記アクションのうちの第1のアクションに基づいて、前記実行主体をノード間で移動させる管理処理と、を含むことを特徴とする。

【0027】請求項7の記録媒体は、請求項6の発明を、当該方法を実現するプログラムを記録した記録媒体の観点から把握したものであって、ソフトウェア上の実行主体がネットワークに接続されたノード上で動作することによって情報処理を行う際に、前記ノードに配置された構成要素へのアクセスに用いるための構成要素の状態を表現する第1の情報を記憶する第1の記憶処理と、前記記憶されている所定の情報を更新する更新処理と、前記実行主体の挙動を定義する第2の情報を記憶するための第2の記憶処理と、前記情報処理に対する要求記述を入力するための入力処理と、前記第1の情報及び前記第2の情報に基づいて、入力された前記要求記述を満たすために実行主体がとるべき行動を表すプランを、アクションの集合として生成する生成処理と、生成された前記プランに基づいてノード上で実行主体の動作を実現する実行処理と、前記生成された前記プランを構成する前記アクションのうちの第1のアクションに基づいて、前記実行主体をノード間で移動させる管理処理と、を含む情報処理方法を実現するコンピュータプログラムを記録したことを特徴とする。

【0028】本発明の他の態様は、請求項1記載の情報処理装置において、前記アクションは必要に応じて、ノード間でのエージェントの移動を実現するgoアクションを含むことを特徴とする。

【0029】請求項1、6、7の発明は、次のような作用を有する。データファイルや機能ライブラリなどの構成要素をネットワーク上の各ノードに分散配置する。各構成要素のドメイン名、ノード名、ディレクトリや呼び

出し手順などは第1の情報として格納しておく。構成要素について、他のノードへの移動など変更があったときは、手動又は自動で第1の情報を更新する。

【0030】情報処理によって実現すべき内容、例えば検索対象とするファイルや検索キーなどは要求記述として入力する。この要求記述を満たすための実行主体の行動はプランとして生成される。プランは、アクション（処理）の集合であり、アクションにはノード移動を行うための第1のアクション（goアクション）が含まれる。

【0031】プラン中の第1のアクションの実行によって実行主体は他のノード（リモートマシン）に転送され、リモートマシン上でアクションの実行や必要なプラン生成が続行される。あるファイルが目的の場合、どのノードにアクセスすべきかなど、具体的な要素は、プラン生成の時点で、第1の情報を参照することによって決定される。

【0032】このように、本発明では、実行主体の挙動はプランによって定まるので、実行主体の動作系列をユーザが記述する必要がなく、情報処理が効率化される。

【0033】また、各構成要素のアクセス先となる具体的なノードなどは、プラン生成の際に第1の情報に基づいて決定される。このため、機能やファイルに変更があっても、その変更が実行主体の挙動に反映される。また、ローカルマシンとリモートマシンの間で情報のやり取りを続ける必要はないので、ネットワークのノード間の接続切断など、予期せぬ状況の変化にも対応して処理が続行される。

【0034】本発明の他の態様は、請求項1記載の情報処理装置において、第1の記憶手段に記憶された第1の情報を更新する更新手段を有してなり、この更新手段は、各ノードにおいて入力された、構成要素に関する変更の内容に基づいて、前記所定の情報を更新する手段を含むことを特徴とする。この態様では、変更の内容を手作業で自由に入力できるので、具体的事情に合わせて入力内容を調整することによって、柔軟な運用が可能となる。

【0035】本発明の他の態様は、請求項1記載の情報処理装置において、前記更新手段は、与えられた条件が成立した場合に、ネットワークに関する情報を参照することによって前記所定の情報を更新する手段を含むことを特徴とする。この態様では、変更の検出と第1の情報の更新が自動的に行われるので、手作業で情報を入力する手間を省くことができ、処理が効率化される。

【0036】請求項2の発明は、請求項1記載の情報処理装置において、前記第2の記憶手段は、実行主体の動作の単位であるアクションの集合を記憶する処理と、前記アクションが実行可能となることを表す事前条件と、実行による状態変化の内容を表す事後条件とを含むアクション情報を記憶する手段と、アクションの実行に関す

る所定の制約条件を記憶する手段と、各アクション間の因果関係を表す情報を記憶する手段と、を備え、前記生成手段は、前記アクションの集合中の所定のアクションに関し、前記アクション情報に基づいて、新たな制約条件又は前記因果関係を表す情報のうち少なくとも一方を、前記第2の記憶手段に記憶させる手段を備えたことを特徴とする。

【0037】請求項3の発明は、請求項2記載の情報処理装置において、前記生成手段は、特定のノードで実行すべきアクションの事前条件を事後条件として実行主体が前記特定のノードへ移動するアクションを前記アクションの集合に追加し、アクションの集合に含まれる各アクションの事前条件のうち、前記因果関係を表す情報において成立する事実として記録されていないものをサブゴールとして順次選択し、選択したサブゴールの事前条件を達成する事後条件を持つアクションをさらに順次選択し、選択された各アクションについて前記新たな制約条件又は前記因果関係を表す情報のうち少なくとも一方を、前記第2の記憶手段に記憶させるように構成されたことを特徴とする。

【0038】本発明の他の態様は、請求項2記載の情報処理装置において、前記生成手段は、選択された各アクションについて、他のアクションの事前条件を損なわないものについては、選択されたアクションを前記因果関係を表す情報に追加するように構成されたことを特徴とする。本発明の他の態様は、請求項2記載の情報処理装置において、前記生成手段は、選択された各アクションについて、他のアクションの事前条件を損なうものであつてかつ当該他のアクションの実行前に当該事前条件が損なわれることを阻止する前記制約条件を追加できるものについては、選択されたアクションを前記因果関係を表す情報に追加するとともに当該制約条件を追加するように構成されたことを特徴とする。

【0039】請求項2、3の発明は、次のような作用を有する。アクションは事前条件が満たされた状態で実行でき、実行の結果、事後条件を発生させる。そこで、目的とする状態を事後条件として発生させるアクションを、因果関係を表す情報によって目的とする状態に接続し、さらに、接続したアクションの事前条件を事後条件として発生させるアクションをさらに接続する。このように実行順序とは逆に遡って接続を繰り返せば、目的とする状態を実現するための動作系列が得られる。このように、本発明によれば、単純な処理を単位としてプラン生成を行えるので、処理が効率化される。

【0040】本発明の他の態様は、請求項2記載の情報処理装置において、前記実行手段は、前記所定の制約条件を満たしつつ、前記第1のアクション以外の第2のアクションを実行した後、該第1のアクションを実行するように構成されたことを特徴とする。この態様では、第1のアクションの実行に先立って、第1のアクション以

外の実行可能なアクションが実行されるので、実行主体を元のノードに再度転送する処理が最小限となり、処理が効率化される。

【0041】請求項4の発明は、請求項1記載の情報処理装置において、未達成ゴールが存在しない場合は、プランの生成が成功で、プランを実行するまでもなく要求記述が達成されたと判定し、未達成ゴールが存在しかつアクション集合が空でない場合は、プランの生成は成功でかつプランの実行が必要と判定する判定手段を有することを特徴とする。請求項4の発明では、プラン生成の結果が判定され、判定結果に応じた処理が選択できるので、処理が効率化される。

【0042】請求項5の発明は、請求項1記載の情報処理方法において、前記実行主体をノード上のプロセスとして設定し、各ノードの前記各管理手段は、転送元となるノードから転送先となるノードへ実行主体を転送するときに、転送元から転送先へ実行主体の受け入れ要求を送信し、受け入れ要求を受信した転送先において、実行主体用のプロセスを設定することによって実行主体を受け入れる準備を行ない、転送先から転送元に準備完了の通知を送信し、準備完了の通知を受信した転送元から前記第1の情報を転送先に送信し、第1の情報を受信した転送先で、受信した第1の情報に基づいて前記プロセスの実行を開始し、転送元において実行主体のプロセスを終了するように構成されたことを特徴とする。

【0043】請求項5の発明では、実行主体がプロセスとして実行されるので、他のプロセスと同時並行的に実行でき、システム効率が向上する。また、各ノードでは、実行主体受け入れ時にプロセスを設定するので、システム資源が有効活用される。

【0044】請求項8の発明は、ソフトウェア上の実行主体であるエージェントがネットワークに接続されたノード上で動作することによって情報処理を行う情報処理装置において、前記ノードに配置された構成要素へのアクセスに用いるための構成要素の状態を表現する第1の情報と、前記実行主体の挙動を定義する第2の情報を記憶するための手段と、前記第1及び第2の情報に基づいて、与えられた目的を達成するためにエージェントが実行すべきプランを作成する手段と、作成されたプランの実行及びエージェントのノード間における移動を実現するための手段と、を有し、移動先のノードにおいても、プランの実行及び必要なプランの作成を行うように構成されたことを特徴とする。

【0045】請求項13の発明は、請求項8の発明を方法の観点から把握したものであって、ソフトウェア上の実行主体であるエージェントがネットワークに接続されたノード上で動作することによって情報処理を行う情報処理方法において、前記ノードに配置された構成要素へのアクセスに用いるための構成要素の状態を表現する第1の情報と、前記実行主体の挙動を定義する第2の情報

を記憶するステップと、前記第1及び第2の情報に基づいて、与えられた目的を達成するためにエージェントが実行すべきプランを作成するステップと、作成されたプランの実行及びエージェントのノード間における移動を実現するためのステップと、を有し、移動先のノードにおいても、プランの実行及び必要なプランの作成を行うことを特徴とする。

【0046】請求項14の発明は、請求項13の発明を、当該発明を実現するコンピュータプログラムを記録した記録媒体の観点から把握したものであって、ソフトウェア上の実行主体であるエージェントがネットワークに接続されたノード上で動作することによって情報処理を行う情報処理方法を実現するコンピュータプログラムを記録した記録媒体において、当該情報処理方法は、前記ノードに配置された構成要素へのアクセスに用いるための構成要素の状態を表現すると、前記実行主体の挙動を定義する第2の情報を記憶するステップと、前記第1及び第2の情報に基づいて、与えられた目的を達成するためにエージェントが実行すべきプランを作成するステップと、作成されたプランの実行及びエージェントのノード間における移動を実現するためのステップと、を有し、移動先のノードにおいても、プランの実行及び必要なプランの作成を行うことを特徴とする。

【0047】請求項8、13、14の発明では、エージェントがプランに基づいてノード間で移動するだけでなく、移動先のノードでもさらにプラン作成が行われる。このため、プランの実行段階や移動先ノードの状態に応じて、情報処理の内容が柔軟に決定され、情報処理が効率化される。

【0048】請求項9の発明は、請求項8記載の情報処理装置において、各ノードに配置された前記各情報が、エージェントの種類ごとに対応する複数のフィールドに区分され、エージェントのプランは、当該エージェントに対応するフィールドの情報に基づいて作成され、ノード間における移動では、エージェントが移動先のノードにおける対応する前記フィールドに移動するように構成されたことを特徴とする。

【0049】請求項15の発明は、請求項13記載の情報処理方法を実現するコンピュータプログラムを記録した記録媒体において、当該情報処理方法は、各ノードに配置された前記各情報が、エージェントの種類ごとに対応する複数のフィールドに区分され、エージェントのプランは、当該エージェントに対応するフィールドの情報に基づいて作成され、ノード間における移動では、エージェントが移動先のノードにおける対応する前記フィールドに移動することを特徴とする。

【0050】請求項9、15の発明では、プラン作成の際、エージェントの種類に対応するフィールドの情報のみが用いられるので、必要な探索処理の量が減り、プラン作成が効率化される。また、ノード間移動の際も、エ

ージェントは対応するフィールドに移動するので、移動前と同様に対応するフィールドの情報を用いてプランの実行やプランの作成が行われる。

【0051】請求項10の発明は、請求項8又は9記載の情報処理装置において、エージェントが、当該エージェントに固有の状態又はアクションに関する情報を持ち、ノード間の移動の際もこれら情報を運搬し、運搬しているこれら情報を移動した先のノードにおける前記第1の情報及び前記第2の情報と合わせてプランニングに用いることを特徴とする。請求項10の発明では、エージェントが固有の情報と共に移動し、移動先のノードでも固有の情報を用いるので、個々のエージェントの状態やアクションに適したプランニングが可能となる。

【0052】請求項11の発明は、請求項8、9又は10記載の情報処理装置において、前記プランを作成する手段は、プランの一部としてエージェントをノード間で移動させる移動命令又はその他の命令を作成する際に、当該命令の前提となる事前条件を当該移動命令又は当該その他の命令に添付するように構成されたことを特徴とする。請求項11の発明では、移動命令に事前条件が根拠として添付される。そして、当該移動命令に基づく移動が失敗する場合は、命令が前提とした事前条件が成立していなかった可能性がある。この事前条件は、移動命令に添付されているので、移動命令を参照すれば容易に判明する。このため、移動失敗の原因が特定され、失敗への対応が容易になる。

【0053】請求項12の発明は、請求項8、9、10又は11記載の情報処理装置において、前記エージェントが、最終的な目的であるゴールを達成するための中間的なサブゴールを作成し、このサブゴールを達成するための子エージェントを生成するように構成されたことを特徴とする。請求項12の発明では、中間的なサブゴールの達成が子エージェントに依託されるので、エージェント毎の情報処理の内容が単純化されることによって、情報処理が効率化される。また、複数のエージェントが並列動作することによって、情報処理が迅速化される。

【0054】本発明の他の態様は、請求項8、9、10、11又は12記載の情報処理において、プランの実行又はエージェントの移動が失敗した場合、プランの実行を中断し、再度プランを作成して実行するように構成されたことを特徴とする。この態様において、望ましくは、プラン中のアクションの実行又はエージェントの移動が失敗した場合、実行又は移動の失敗を回復するためのサブゴールを生成し、当該サブゴールの実行終了後に前記プランの実行を再開するための情報を保存し、前記サブゴールに基づいてプランの作成及びプランの実行を行い、サブゴールに基づくプランの実行後に、保存した前記情報を用いて元のプランの実行を継続するように構成する。このような態様では、プランの実行や移動が失敗しても、再度プランが作成されるので、情報処理が円

滑に継続される。

【0055】本発明の他の態様は、請求項8、9、10、11又は12記載の情報処理装置において、プランの実行又はプランに基づく移動が失敗した場合に、失敗の原因となった情報を修正する手段を有することを特徴とする。本発明の他の態様は、請求項13記載の情報処理方法において、プランの実行又はプランに基づく移動が失敗した場合に、失敗の原因となった情報を修正するステップを有することを特徴とする。これらの態様では、失敗の基礎となった情報が修正されるので、それ以降の失敗が減少し、処理が効率化される。

【0056】

【発明の実施の形態】以下、図面を参照しながら本発明の実施の形態（以下「実施形態」という）について説明する。なお、以下の各実施形態は、複数のノードを含むネットワークによる分散環境を前提とする。ここで、

「ネットワーク」は、計算機と計算機を接続するもの一般を意味し、具体的には、インターネット、電話回線、等である。「ノード」は、エージェントの移動の単位である。基本的には、一台のホストすなわち一台の計算機上に1つのノードを想定するが、複数のノードが存在していてもよい。「分散環境」は、ネットワークで接続されたコンピュータが複数存在し、それぞれのコンピュータが持つ情報が異なっている環境である。

【0057】また、下記の各実施形態は、コンピュータ上においてプログラムによってCPUを制御することによって実現され、この場合の具体的な実現態様は種々考えられる。このため、以下の説明では、本装置の各機能に対応する「～部」などの仮想的回路ブロックとして本装置を説明する。なお、CPUは、プログラムで指定されたとおりに、コンピュータの各種ハードウェア資源を利用しながら、前記各仮想的回路ブロックの作用を実現する。

【0058】ハードウェア資源の典型例として、CPUには、バス及び入出力制御回路を介して、RAMなどの記憶素子からなるメモリ、ハードディスクドライブなどの補助記憶装置、入力装置としてマウスやキーボード、出力装置として表示装置やプリンタを接続することが考えられる。また、コンピュータは、ネットワーク接続機器を介してネットワークに接続される。但し、これらハードウェア資源は例示に過ぎず、情報の記憶・入力・出力などの目的を達成できる他の各種装置を用いることもできる。

【0059】1. 第1実施形態

以下、本発明の実施の形態（以下「実施形態」という）である情報処理装置（請求項1～5に相当するもの）及び情報処理方法（請求項6に相当するもの）について、図面を参照して説明する。

【0060】なお、本実施形態の目的は、ソフトウェアの構成要素の変化に柔軟に対応し、かつ、回線障害に対

する耐障害性に優れた情報処理装置及び情報処理方法を提供することである。また、本実施形態の他の目的は、柔軟な運用が可能な情報処理装置及び情報処理方法を提供することである。また、本実施形態の他の目的は、処理の効率に優れた情報処理装置及び情報処理方法を提供することである。

【0061】(1) 構成

【ハードウェアの構成】図1は、本実施形態の構成を示す機能ブロック図である。この図に示すように、第1実施形態では、ノードL、Rを含む複数のノードがネットワークNで接続されている。ここでは、要求記述が入力され、処理が開始されるノードLをローカルマシン、ローカルマシンからエージェントが移動する先のノードRをリモートマシンと呼ぶ。そして、情報処理に用いるソフトウェアの構成要素はこれら複数のノードに分散して配置され、ソフトウェア上の実行主体であるエージェントがいずれかの前記ノード上で活動することによって情報処理を行う。

【0062】各ノードL、Rは、構成要素へのアクセスに用いるローカル情報（前記第1の情報に相当するもの）を格納するローカル情報記憶手段1（1L及び1R、以下同様。前記第1の記憶手段に相当するもの。）と、前記格納されているローカル情報を更新する更新手段2と、を有する。また、各ノードL、Rは、エージェント情報（前記第2の情報に相当するもの）を格納するためのエージェント情報記憶手段3と（前記第2の記憶手段に相当するもの）、各種入出力を行うための入出力手段4（情報処理に対する要求記述を入力するための前記入力手段に相当する）と、を有する。なお、エージェント情報は、エージェントの持つ挙動に関する情報であり、エージェント自身の挙動によって順次更新されるものである。

【0063】また、各ノードL、Rは、入力された前記要求記述を満たすためにエージェントがとるべき行動を表すプランを、エージェント情報及びローカル情報に基づいて、複数のアクションの集合として生成するプラン生成手段5（前記生成手段に相当するもの）を有する。プラン生成手段5によって生成されるプランに含まれるアクションは必要に応じて、ノード間でのエージェントの移動を実現するgoアクション（前記第1のアクションに相当するもの）を含む。

【0064】また、各ノードL、Rは、生成された前記プラン中の各アクションに基づいて各ノードL、R上でエージェントの動作を実現するプラン実行手段6（前記実行手段に相当するもの）と、プラン中のgoアクションに基づいて、エージェントをノード間で移動させるエージェント管理部7（前記管理手段に相当するもの）と、を有する。なお、エージェント管理部7は、生成されたプランについて判定を行う判定手段（請求項4）としての機能を有している。

【0065】また、本実施形態で用いられる主な情報を次に説明する。

【0066】【エージェント情報】…第2の情報

エージェント情報は、エージェントの挙動を制御する情報及びこの挙動によって操作される各種の情報である。エージェント情報として、具体的には、アクションの集合（以下「アクション集合」という）と、集合に含まれる各アクションに関する情報（以下「アクション情報」という）と、安全性に関する制約条件（以下「安全性条件」という）と、因果リンク（前記因果関係を表す情報に相当するもの）とを挙げることができる。まず、アクション集合は、生成中のプランを構成するアクションの集合をリストで表現したものであり、アクションはエージェントがとりうる動作の単位である。

【0067】各アクションについては、アクション情報が存在する。アクション情報は、アクションの名称と引数（パラメタ）を表すアクション記述、アクションが実行可能となるための前提条件を表す事前条件、実行による状態変化の内容を表す事後条件を含み、これらはいずれも項又は項のリストで表される。

【0068】安全性条件は、アクション集合中の各アクションについて、アクション間の実行順序の制限を表す条件であり、アクション記述の対のリストで表現される。すなわち、安全性条件を構成する各対は、1番目の要素は2番目の要素よりも前に実行すべきである、という制約をあらわす。アクション集合のリスト中の順序と実際の実行順序とは無関係であり、実行順序はこの安全性条件の制約を受けるのみである。

【0069】因果リンクはアクション集合中の各アクションに対し、1つのアクションの実行により成立した事実により、別のアクションが実行可能となる、といった因果関係を示す情報で、アクション記述、そのアクションにより成立する事実を表す項、およびその事実の成立により実行可能となるアクション記述の3つの組のリストで表現される。

【0070】アクション情報の例を次に示す。

アクション記述：grep (File, Keyword)

事前条件：[file-exists (File)]

事後条件：[add (include (File, Keyword))]

この例において、アクション記述“grep (File, Keyword)”は、“grep”という名称の動作で、“File”という名称のファイルについて、文字列“Keyword”が含まれているかどうか検索し、含まれていた場合に事後条件を実現する、という動作を表す。事前条件は、そのアクションを実行するときに成立していなければならない条件を表す。この例の事前条件“file-exists (File)”は、前述の状態記述と同じ書式で、“File”という名称のファイルが存在するという事実を表す。

【0071】事後条件は、一般にはアクションの実行によって発生する事実を表し、この例では、文字列がファ

イル含まれていたときに限って発生する。この例の事後条件"add(include(File, Keyword))"の中で、"include(File, Keyword)"は"File"という名称のファイルが、文字列Keywordを含む、という事実を表す。"add 0"は、括弧内に記述された事実（ここでは"include(File, Keyword)"をエージェントの知識として追加する動作を表す。なお、逆に知識を削除する動作は"del 0"のように表す。

【0072】〔ローカル情報〕…第1の情報

ローカル情報記憶手段2には、ローカル情報が格納される。本実施形態におけるローカル情報は、構成要素にアクセスするための情報で、項と呼ばれる記号列で表現したものである。本実施形態では、以下のような例で示される状態記述が、ローカル情報記憶手段2に格納されているとする。なお、状態記述は、ソフトウェアの構成要素に関する何らかの事実を表す情報である。

```
maybe(file-exists(file1) at node1)
```

この例において、"maybe 0"は、括弧内に記述された事実が未確認であることを表す。また、"file-exists(file1)"はfile1なるファイル名を持つファイルが存在することを示す。また、そのあとの"at node1"は、"node1"なるノード名を持つマシンにおいて、前述の事実（ファイルfile1の存在）が成立することを示す。状態記述が"maybe 0"であるときは、前述の事実（マシンnode1におけるファイルfile1の存在）の正否を、マシンnode1に移動して確認する必要がある。ここでは、ノード名node1は、リモートマシンRのノード名であるものとする。

【0073】なお、このようなローカル情報の更新は、例えば、システムの管理者が、各マシンにおいて構成要素が変更された際に、手動で情報を更新することによって行うことができる。このようにすれば、具体的な事情に合わせて入力内容を調整することによって、柔軟な運用が可能となる。

【0074】また、ローカル情報は、プログラムによって、ネットワーク中の構成を表すファイルリストなどを参照することによって自動的に更新することもできる。このようにすれば、変更の検出とローカル情報の更新が自動的に行われるので、手作業で情報を入力する手間を省くことができ、処理が効率化される。

【0075】以上のようなエージェント情報及びローカル情報は、上記のような文字列テキストの形式で格納しておくことが考えられるが、必要に応じて予約語をコードに置き換えるなど、他の内部形式で格納しておくこともできる。

【0076】（2）作用及び効果

上記のような構成を有する本実施形態において、情報処理は次のように行われる。

【0077】〔要求記述の入力〕図3に、本実施形態による情報処理の処理手順のフローチャートを示す。ま

ず、情報処理によって達成すべき状態の記述を、ユーザが入出力手段1から要求記述として入力する（ステップ201）。入力された要求記述は、エージェント情報記憶手段3に格納される。ここで、要求記述は1つもしくは複数の項から構成される。要求記述の一例を次に示す。

```
file-exists(File) at Node
```

```
include-keyword(File, patent)
```

この例の1行目"file-exists(File) at Node"はNodeというノード名のマシン上において、Fileという名称のファイルを見つける、という要求を表す。また2行目は、Fileがpatentというキーワードを含んでいることを要求している。したがって、本要求記述は、patentというキーワードを含むファイルをネットワークから探して、そのファイル名とノード名を特定する、という要求を表すこととなる。

【0078】〔初期化〕次に、エージェント管理部7は初期化処理を行う（ステップ202）。すなわち、エージェント情報記憶手段3内のアクション集合、安全性条件、および因果リンクとして、それぞれ下記のような情報を区別して格納する。

アクション集合：[*start*, *end*]

安全性条件：[(*start*, *end*)]

因果リンク：[]（空リスト）

さらに、エージェント情報記憶手段5に対し、次の情報をアクション情報として追加・格納する。まず、初期状態のうち、"maybe 0"で表された条件（maybe条件と呼ぶ）以外のもののリストを、"*start*"というアクション名を持つアクションの事後条件とする。

【0079】また、初期状態のうちの"maybe(cond at node)"（condおよびnodeは、それぞれ具体的な条件およびノード名を意味する）の形の条件に対し、次のようなアクション情報を追加する。

アクション名：go(node)

事前条件：[]

事後条件：[add(cond at node)]

すなわち、maybeを取り除いた条件を事後条件とするgoアクションを定義する。これは、特定のノードで実行すべきアクションの事前条件を事後条件としてエージェントが前記特定のノードへ移動するアクションを前記アクション集合に追加することを意味する。

【0080】たとえば、本実施形態においては、次のようなアクション情報を追加する。

【0081】アクション名：*start*

事前条件：[]

事後条件：[]

アクション名：go(node1)

事前条件：[]

事後条件：[add(file-exists(File) at node1)]

アクション名：*end*

事前条件：[(file_exists(File) at Node),
include-keyword(File, patent)]
(要求記述と同じ)

事後条件：[]

【0082】[プランの生成] 続いて、プラン生成手段5によって、要求記述を満足するためのプラン生成が行われる(ステップ203、請求項2、3)。プラン生成は、ローカル情報を参照しながら、エージェント情報を更新することによって行われ、更新されたエージェント情報がプランとなる。なお、プラン生成の手順は、エージェントの転送に関する部分を除いて、従来から知られるプランニング技術(参考文献1：大須賀節雄編「人工知能大事典」979ページから988ページ)によって行う。このプランニングは、ロボットなど外界を変更する能力を持つ行為主体について、与えられた目標を達成するための行動プランを作成することである。プランニングのために与える入力、外界の初期状態、外界を変化させる様々なアクション(行為)及び一つ又は複数の目標である。

【0083】本実施形態におけるプランニングの手順を、図3のフローチャートに示す。まず、未達成ゴールの有無を調べる(ステップ301)、ここで未達成ゴールというのは、アクション集合を構成する各アクションの事前条件のうち、因果リンクにおいて成立する事実として記録されていない条件を項で表したもののリストである。本実施形態では、最初因果リンクが空であるので、アクション"*end*"の2つの事前条件が未達成ゴールとなる。

【0084】未達成ゴールが存在しない場合はプラン生成を終了するが、存在する場合は、未達成ゴールから1つサブゴールとして選択する(ステップ302)。本実施形態では、たとえば"file-exists(File) at Node"をサブゴールとして選択する。次にステップ302において、選択したサブゴールに対し、事後条件において該サブゴールを達成するようなアクションを列挙する。

【0085】ここで条件がサブゴールを達成するとは、該条件の変数に適当な項を代入することにより、該サブゴールと一致させることができる、ということであり、さらに具体的には、選択したサブゴールの事前条件と変数以外の要素が一致する事後条件を持つアクションを意味する。たとえば本実施形態では、アクション"go (node l)"のみを列挙する。

【0086】そして、アクションが1つでも列挙できたかどうかを判定し(ステップ304)、できなかった場合には、いわゆるバックトラック処理として、サブゴール選択をやり直す(ステップ302)。

【0087】そして、列挙したアクションから1つ選択し(ステップ305)、選択したアクションについて、他のアクションの事前条件を損なうものについては、選

択されたアクションをサブゴールの条件に合わせて因果リンクに追加する。また、他のアクションの事前条件を損なうものであつてかつ当該他のアクションの実行前に当該事前条件が損なわれることを阻止する安全性条件を追加できるものについては、選択されたアクションをサブゴールの条件に合わせて因果リンクに追加するとともに当該安全性条件を追加する。

【0088】これらの処理は、具体的には次のように行う。まず、該アクションのアクション情報に基づき因果リンク及び未達成ゴールを更新する(ステップ306)。因果リンクの更新は、前述の代入を、該アクション情報の要素(アクション記述、事前・事後条件)に施し、アクション記述、サブゴール、およびサブゴールを事前条件とするアクションのアクション記述の組を、因果リンクに追加することにより行う。そして同時に、該アクション情報の事前条件を、新たに未達成ゴールに追加する。たとえば本実施形態では、アクション"go (node l)"を選択し、その結果、因果リンク、および未達成ゴールを以下のように更新する。

因果リンク：[(go (node l), file-exists (File) at node l, *end*)]

未達成ゴール：[include-keyword (File, patent)]

【0089】[脅威の検出] 次にステップ307において、選択したアクションによって、他の因果リンクの条件、すなわち他のアクションが前提とする状況を損なうかどうかを調べる(ステップ307)。損なう場合、これを脅威と呼ぶ。本実施形態では、選択したgoアクションは、条件を損なわない。すなわち、goアクションが一般に条件を損なうわけではなく、この場合はノード移動によって不可能になるアクションがないからである。

【0090】脅威が存在する場合、この脅威を回避するための安全性条件の更新が可能かどうかを判定する(ステップ308)。ここで、安全性条件の更新が可能な場合は、次のような場合である。

【0091】例えば、因果リンクに追加した組(アクション1、条件、アクション2)に対し、対(アクション1、アクション2)を安全性条件に矛盾なく追加でき、かつ脅威が存在する場合は、すなわち他の因果リンク(アクション3、条件2、アクション4)に対し、アクション1が条件2を損なう場合に、対(アクション3、アクション1)もしくは(アクション4、アクション1)を安全性条件に矛盾なく追加できる場合である。

【0092】そして、安全性条件更新が不可能な場合には、更新した因果リンクおよび未達成ゴールの状態を更新前に戻し(ステップ310)、いわゆるバックトラックとして、アクションの選択からの処理をやり直す。安全性条件更新が可能な場合は、ステップ309において上述した安全性条件の更新(対の追加)を実行する。

【0093】本実施形態では、対"(go (node l), *end*)"を矛盾なく安全性条件に追加できるので、更新の結果安

全性条件は“(go (node1), *end*), (*start*, *end*))”となる。その後再びステップ301からの処理を繰り返される。このように、必要なローカル情報を参照しながら処理を繰り返すことによって、全ての未達成ゴールの一つ一つについて、当該未達成ゴールに対応する全てのアクションについて、因果リンクへの追加が試みられることになる。

【0094】すなわち、アクションは事前条件が満たされた状態で実行でき、実行の結果、事後条件を発生させる。そこで、目的とする状態を事後条件として発生させ

```
アクション集合: [grep (File, patent), go (node1)]
安全性条件: [(grep (File, patent), *end*),
              (go (node1), *end*), (*start*, *end*)]
因果リンク: [(grep (File, patent),
              include (File, patent), *end*),
              (go (node1), file-exists (File) at node1, *end*)]
未達成ゴールリスト: [file-exists (File)]
```

このエージェント情報の内容を表すツリー図（プラン木）を図4に示す。

【0096】【プランの判定】以上のようにプランが生成されると（図2、ステップ203）、プランの生成に関して判定が行われる（ステップ204、請求項4）。このとき、未達成ゴールが存在しかつアクション集合が空の場合はプランの生成は失敗と判定される。また、未達成ゴールが存在しない場合は、生成が成功で、プランを実行するまでもなく要求記述が達成されたと判定される。また、未達成ゴールが存在しかつアクション集合が空でない場合は、プランの生成は成功でかつプランの実行が必要と判定される。

【0097】プランの生成が成功で要求記述が達成されている場合と、プランの生成そのものが失敗の場合は、エージェントは入出力手段4よりユーザにその旨を出力し（ステップ205）、全手順を終了する。なお、エージェントが当初のローカルマシンLから移動し、リモートマシンRにおいて要求記述が達成され又はプランの生成に失敗したときは、エージェントはローカルマシンLに戻ってその旨の情報を出力する。このように、本実施形態では、プラン生成の結果が判定され、判定結果に応じた処理が選択できるので、処理が効率化される。

【0098】【プランの実行】プランの実行では、プラン実行手段6が、goアクション以外の実行可能なアクションを探す（ステップ206）。実行可能なアクションは、アクション集合中のアクションのうち、その時点の状態によって事前条件が満足され、かつ安全性条件に関しても他のアクションより後で実行する必要がないものである。

【0099】goアクション以外の実行可能なアクションがあれば（ステップ207）、そのアクションが実行される（ステップ208）、再度ステップ206からの手順が実行される。goアクション以外の実行可能なアクション

るアクションを、因果リンクによって目的とする状態に接続し、さらに、接続したアクションの事前条件を事後条件として発生させるアクションをさらに接続する。このように実行順序とは逆に遡って接続を繰り返せば、目的とする状態を実現するための動作系列が得られる。このように、本発明によれば、単純な処理を単位としてプラン生成を行えるので、処理が効率化される。

【0095】【エージェント情報の更新】本実施形態においては、繰り返し手順終了後に、エージェント情報記憶手段3に格納された情報が以下のように変化する。

ンがないときは、goアクションが一つ実行される（ステップ209）。

【0100】このように、本実施形態では、goアクションの実行に先立って、goアクション以外の実行可能なアクションが実行されるので、エージェントを元のノードに再度転送する処理が最小限となり、処理が効率化される。

【0101】本実施形態の例では、プランの実行が開始された時点で、goアクション以外のアクションとして“grep (File, patent)”が存在する。しかし、この時点ではこのアクションは、事前条件が未達成であるためにこのアクションは実行できず、結果的にgoアクション以外の実行可能なアクションがない。このため、アクション“go (node1)”が実行される。

【0102】goアクションが解釈実行されると、プラン実行手段6がその旨をエージェント管理部7に通知し、エージェント管理部7は次のようなエージェント転送処理を行う。

【0103】【エージェントの転送】エージェントを転送するときは、ローカルマシン側のエージェント管理部7L（以下「ローカル側」という）とリモートマシン側のエージェント管理部7R（以下「リモート側」という）との間で、ネットワークNを通じ、次のような手順が実行される（請求項5）。図5はエージェントを転送する手順を示すフローチャートである。

【0104】まず、ローカル側からリモート側へ、エージェントの受け入れ要求が送信される（ステップ501）。リモート側は、要求を受信すると（ステップ502）、エージェント用のプロセスを設定することによってエージェントを受け入れる準備を行なった後（ステップ503）、ローカル側に準備完了の通知を送信する（ステップ504）。

【0105】ローカル側は、準備完了の通知を受信する

と(ステップ505)、エージェント情報記憶手段3内のエージェント情報を読み出し、リモート側に送信する(ステップ506)。(転送されるエージェント情報は、goアクションが解釈実行された時点におけるエージェントの内部状態を含む。)リモート側は、エージェント情報を受信すると(ステップ507)、受信したエージェント情報をエージェント情報記憶手段3に格納し(ステップ508)、プラン実行手段6にその旨を通知することによって解釈実行を開始させ(ステップ509)、エージェント受け入れが成功した旨の通知をローカル側に送信する(ステップ510)。

【0106】ローカル側は、成功の通知を受信すると(ステップ511)、エージェントのプロセスを消去し(ステップ512)、不要になったメモリなどの資源を開放することによって、エージェントの転送を終了する。

【0107】本実施形態の例では、転送後のリモートマシンnode1において、ローカル情報記憶手段1に、以下のような情報が格納されているものとする。

file-exists(file)

さらに、事実マシンnode1にはfile1なるファイルが存在し、しかも文字列patentを含んでいるものとする。この場合、エージェントはマシンnode1に移動後、改めて初期化からプラン生成の手続きを実行する。その結果、エージェント情報記憶手段に格納された情報は、以下のように変化する。

アクション集合:[grep(file1,patent)]

安全性条件:[(grep(file1,patent),*end*),(*start*,*end*)]

因果リンク:[

(*start*,file-exists(file),grep(file1,patent)),(grep(file1,patent),include(file1,patent),*end*)]

未達成ゴールリスト:[

これにより、プラン生成成功となり、アクションgrep(file1,patent)の実行によりプラン実行成功となるので、エージェントはローカルマシンに戻ってユーザにその旨の情報を出力する。図6に実行結果の出力例を示す。この例では、表示用のウインドウ中に、最終的なゴール500としての要求記述と共に、実行結果501が表示されている。

【0108】また、仮にファイルfile1が他のマシンに移されている、代わりに"maybe(file-exists(file2)at node2)"などの情報がローカル情報記憶手段に格納されていれば、再プランニングによりマシンnode2にエージェントが移動して、目的を達成することができるので、環境の変化にも柔軟に対応しうる。

【0109】以上のように、本実施形態によれば、エージェントの挙動はプランによって定まるので、エージェントの動作系列をユーザが記述する必要がなく、情報処理が効率化される。

【0110】特に、各構成要素のアクセス先となる具体的なノードなどは、プラン生成の際にローカル情報に基づいて決定される。このため、機能やファイルに変更があっても、その変更がエージェントの挙動に反映される。また、ローカルマシンとリモートマシンの間で情報のやり取りを続ける必要はないので、ネットワークのノード間の接続切断など、予期せぬ状況の変化にも対応して処理が継続される。

【0111】2. 第2実施形態

第2実施形態は、第1実施形態と略同様(図1)の構成を有する情報処理装置について、その構成及び作用をより具体的に示すもので、特に、移動先ノードにおいても、プラン実行の失敗時など必要に応じて、再度のプランニングが行われるものである。

【0112】(1) 構成

(1-1) 全体構成

第2実施形態は、第1実施形態と同様、ネットワークに接続された複数のノード上で、ソフトウェア上の実行主体であるエージェントが動作することによって情報処理を行う情報処理装置である。図7は、第2実施形態において、ネットワークNに接続された各ノードの構成を示す機能ブロック図である。この図に示すように、各ノードは、フィールド知識ベース11と、ノード知識ベース12と、プランナ13と、実行器14と、ノードマネージャ15と、更新器16とを有する。

【0113】すなわち、まず、各ノードは、ノードに配置された構成要素へのアクセスに用いるための第1の情報としてローカル情報を有する。データベースであるフィールド知識ベース11とノード知識ベース12は、まず、この第1の情報を記憶するもので、第1実施形態におけるローカル情報記憶手段に対応する。すなわち、ローカル情報は、エージェントの種類ごとに対応する複数のフィールドに区分された部分と、ノード固有の部分とがあり、それぞれ、フィールド知識ベース11とノード知識ベース12に記憶されている。フィールドは、特定の種類すなわち目的や分野のエージェントのみが利用する情報のグループである。

【0114】なお、第2実施形態では、各知識ベース11、12には、前記のローカル情報に加え、エージェントがとりうるアクションに関する第2の情報も記憶されている。プランナ13は、これら第1及び第2の情報に基づいて、与えられた目的を達成するためにエージェント17が実行すべきプランの作成(プランニング)を行う部分で、第1実施形態におけるプラン生成手段に対応する。

【0115】実行器14は、作成されたプランの実行を行う部分であり、第1実施形態におけるプラン実行手段に対応する。また、ノードマネージャ15は、エージェント17の生成/消滅及びノード間における移動を実現するための部分で、第1実施形態におけるエージェント

管理部に対応する。なお、各ノードのプランナ3及び実行器14は、他のノードから移動してきたエージェントについても、プランの実行及び必要なプランの作成を行う（請求項8、13）。

【0116】なお、第1の情報で表される情報は、ノード上のソフトウェア要素に関するもので、具体的には、そのノードの状態や、ノードに存在するファイル、データベース、保有するソフトウェア等である。また、アクションに関する第2の情報はノード毎に定義される。

【0117】プランナ13は、エージェントのプランを作成する際、当該エージェントに対応するフィールドの情報に基づいて作成するように構成されている。また、ノードマネージャ15は、ノード間における移動の際、エージェントが移動先のノードにおける対応するフィールドに移動するように構成されている（請求項9）。なお、エージェントを移動先の所定のフィールドに移動させるには、プラン中に含める移動命令において、命令のパラメータなど所定の形式でフィールドを指定すればよい。

【0118】また、更新器16は、プランニング、プランの実行又はプランに基づく移動が失敗した場合に、失敗の原因となった情報を修正するための部分で、第1実施形態における更新手段に対応する。

【0119】また、ノードは、利用者インターフェースとしてGUI（グラフィカルユーザインターフェース）アプリケーション18を用いる。ここで、利用者インターフェースは、ユーザがエージェントの状態を直接監視するために用いるソフトウェアで、第1実施形態における入出力手段に対応する。エージェントはソフトウェア上の実体であるため、ユーザがその状態を参照したり、制御するためには、ユーザインターフェースを有するソフトウェアが必要となり、このソフトウェアとして、GUIアプリケーション18を用いるものである。また、ノード毎のメモリなど資源の管理に関する情報は、ノード管理情報記憶手段19に記憶される。

【0120】ここで、第2実施形態において例として用いる具体的なノード構成の想定を図8に示す。この例においては、ネットワークNに、node0、node1、node2という3個のノードXが接続された情報処理装置を想定する。各ノードXには、makeという名称のフィールドFが存在し、その他のフィールドは省略する。すなわち、第2実施形態におけるノード構成の例は、例えば、複数の人手によるソフトウェア開発等において、ネットワーク上に分散して作成される複数のソースファイルを、ある必要な時点において特定のノードに集積することを想定したもので、特に、その中の一部の処理を取り出して説明するものである。

【0121】なお、図8に示すように、エージェントAは、当該エージェントAに固有の状態又はアクションに関する情報を持ち、この情報はエージェント内の知識ベ

ースDAに記憶されている。エージェントAは、ノード間の移動の際もこれら情報を運搬し、運搬しているこれら情報をプランニングに用いる（請求項10）。

【0122】（1-2）エージェントの概略的構成
エージェントは、システムの利用者、又は本システムを利用する別のソフトウェアが、ゴールを与えることによって、ノード上に生成されるプロセスである。なお、ノードとは、エージェントの移動の単位を指しており、抽象的な概念である。1台のホストに対して、1つのノードが存在することを通常の想定とするが、1台のホストに対して複数のノードが存在することもありうる。ユーザは、ノードに対し、ネットワークにおいて唯一となるノード名を与えておく。この場合、ノード名としては、URL等の既存のコンピュータネットワークのネーミング手法によってつけられた名称と同じ名称を与えることが可能である。また、上記の「プロセス」は、計算機上でソフトウェアを実行する単位であり、オペレーティングシステムによって管理される。

【0123】図9は、エージェントを、Plangentのモバイル・エージェントとして構成する場合の環境を示すもので、ノードがフィールドに区分されている状態を示す概念図である。フィールドに区分されるのは実際にはノード毎に格納されている第1の情報や第2の情報であるが、これらの情報はプラン作成やプラン実行などエージェントの活動の基礎すなわち活動の場としての役割を持つ。したがって、ノード上の情報が区分されることによって、当該ノード上には、エージェントの活動の「場」であるフィールドが複数存在することになる。

【0124】図に示す環境構成の要素は、ホストH上に常駐するノードX、ノードX内を分割するフィールドF、ネットワークNを介してノードXを移動するエージェントAなどである。

【0125】このようなエージェントの目的は、情報処理の目的として所定の表現形式で与えられたゴールを満たすことにある。ゴールの一例は、他のどこかのノードにあるはずの特定のファイルを探して、コピーを持ち帰ることである。このために、エージェントは、与えられたゴールを満たすプランを生成し、実行する。エージェントは、ゴールの達成のために必要であれば、他のノードに移動することが可能である。エージェントによるこれらプラン作成やプラン実行、ノード間の移動などの処理は、具体的にはプランナ13、実行器14、ノードマネージャ15などによって実現される。

【0126】そして、生成されたエージェントは、最終的にゴールを達成するか、ゴール達成不可能となるか、いずれかの状態に至り、消滅する。ゴールを達成した場合をエージェントの正常終了と呼び、ゴール達成不可能となった場合をエージェントの完全失敗と呼ぶ。消滅の際に、終了時処理が行われる。

【0127】図10に、エージェントの活動における複

数のフェーズを説明する状態遷移図を示す。すなわち、生成されたエージェントは、その後、プランニング・フェーズP、実行フェーズE、移動フェーズMの3つのフェーズを繰り返しながら、情報処理を行う。

【0128】(1-3) エージェントの論理構造
次に、第2実施形態におけるエージェントの論理的な構造を図11に示す。すなわち、エージェントは、エージェント名、対応するフィールドのフィールド名、所有ファイルなどのデータを有する。なお、所有ファイルはエージェントが自己の一部として所有するファイルで、他のノードへ移動したエージェントがファイルのコピーを所有ファイルとして元のノードへ持ち帰ることによって、例えば、先に例として示したファイルを収集する目的において、効果的な情報収集が可能となる。また、エージェントは、これらに加えて、次のような他の構成要素を有する。

【0129】まず、ゴールスタックSは、プランニングを階層的に実施するために、実行途上のゴールを積み上げたスタックである。ゴールスタックは、ゴール・スクリプト・セットの集合として保持される。ゴール・スクリプト・セットは、最終的なゴールやサブゴールと、それら各ゴールを達成するためのプランのスクリプトのセットであり、エージェントがその時点において所有している全てのスクリプトを含む。

【0130】また、エージェント知識ベースDAは、エージェントがプランニングにおいてノードの知識ベース及びフィールドの知識ベースとともに使用する知識ベースである。また、図11の「その他」の要素としては、例えば実行情報が挙げられる。実行情報は、エージェント名、フィールド名、スクリプトにおける実行位置、およびスクリプトで用いられる変数の値などの情報である。また、「その他」の情報としては、終了時処理の指定等の情報も考えられる。これらの情報を表す具体的なデータ構造は、エージェントのフェーズによって異なっており、例えば、移動フェーズ中にはネットワーク上の通信内容として、実行フェーズ中には記憶媒体におけるファイル構造として、プランニングフェーズではプログラム上のデータ構造として計算機上のメモリ上に記憶される。

【0131】ゴールをスタック構造で表すのは、ユーザが与えたゴールが、一般にプランニングによって階層的な構造をとるためである。すなわち、新たに作成されるサブゴールは、エージェントの内部においてスタックの形式により階層的に保有される必要がある。そのゴールスタックを構成するゴール・スクリプト・セットのデータ構造を図12に示す。

【0132】この図において、「ゴール」は、エージェントを利用するユーザ、又は別のソフトウェアが与える要求である。任意のゴール・スクリプト・セットについてみれば、ゴールは、サブゴールである場合も考えられ

る。すなわち、「サブゴール」は、最初エージェントに与えられたゴールを満たすために必要な条件を分解したゴールである。

【0133】第2実施形態で作成されるサブゴールには、失敗時サブゴールとプランニング時サブゴールの2通りが考えられる。失敗時サブゴールとは、エージェントによるスクリプトの実行中に、特定のコマンドの実行が失敗した場合に、別の態様でそのコマンドの実行を試行するために生成するサブゴールである。この場合、例えば、ファイルの所在地として第1候補のノードでファイル発見に失敗した場合、第2候補のノードでファイルを発見することがサブゴールである。

【0134】プランニング時サブゴールとは、通常のプランニングの結果としてサブゴールがスクリプトに含まれるものである。例えば、ファイルの発見と持ち帰りがゴールの場合、まず、ファイルの発見や、発見のために他のノードへ移動することがサブゴールとなる。

【0135】サブゴールは、さらにサブゴールを持つことが可能であり、ゴールは全体として階層的な構造を有する。「ゴールスタック」は、実行中のエージェントが保有するもので、階層的ゴール構造を管理するスタックである。エージェントは、その生成時に与えられた最終的なゴールとサブゴールとを階層的に保有する。

【0136】(1-4) ノード・マネージャ
上記のようなエージェントの管理を行う部分がノード・マネージャ15である。ノード・マネージャは、ノードを管理するモジュールで、特に、エージェントの生成及びノード間におけるエージェントの移動などの処理を行う。このような処理を行う際に、ノードマネージャは、他のノードのノードマネージャと必要に応じて通信を行う。この通信におけるインタフェースを以下、ノード間インタフェースと呼ぶが、第2実施形態では、インターネット・プロトコルスイートにおけるTCP によるポートを用いて実装し、名称をノード・ポートと呼称するものとする。各ノードマネージャは、それぞれのポートにおいてサーバとして機能する。

【0137】また、ノードマネージャは、自らが生成したエージェントとも通信を行う。この通信におけるインタフェースを以下、ノード=エージェント間インタフェースと呼ぶ。第2実施形態では、ノード・マネージャのノード・ポートを用いて、ノード=エージェント間インタフェースを実装し、ノード・マネージャ側をサーバ、エージェント・プロセス側をクライアントとする。

【0138】図13は、ノードマネージャ15を中心として、ノード=エージェント間通信及びノード=ノード間通信を実現するための構成例を示す図である。各ノードには、エージェントの状態を出力するためのソフトウェアモジュールであるモニターMNが設けられ、ノードマネージャ15は、モニターMNに対して、表示の指示となるモニター・スクリプトを送信する。なお、第2実

施形態におけるポートの名称をモニター・ポートMPと呼称する。ノード・マネージャ側がサーバ・ポートとし、モニターMN側をクライアント・ポートとする。以下、特に断らなければ、単にノードポートと呼称した場合は、同一ノード内のノード・ポートNPを指す。各通

信ポートにおける通信プロトコルは、定められた所定のデリミタによって区切られる下記の語より構成される。

【0139】まず、実施例によるノード間インタフェースのプロトコルを表に示す。

【表1】

先頭文字列	意味	引数
AS	新規エージェント起動	ゴール, フィールド名, 終了時処理方式
MF	他ノードからのエージェントの移動	エージェント名, フィールド名, ゴールスタック, スクリプト, エージェント知識ベース, 実行情報

【0140】また、実施例によるノード=エージェント間インタフェースのプロトコルを表に示す。

【表2】

先頭文字列	意味	引数
AE	エージェントの終了通知	エージェント名, ステータス(成功・失敗)
ST	エージェントのステータス通知	エージェント名, ステータス名
MT	他ノードへのエージェントの移動	行先ノード名, エージェント名

【0141】ノード・マネージャは、モニターによるエージェントの監視のために、モニター・ポートよりモニタースクリプトを出力する。この通信ポートについては、クライアントであるモニターよりセッションがサーバであるノードマネージャに張られる。モニターは、モ

ニタースクリプトを受け取り、ユーザ・インタフェースの更新を行うことができる。モニタースクリプトの出力内容は、基本的には、ノード・ポートに対するアクセスに応じて、次に示すような形式を定めている。

【表3】

ノード・ポートへの入力語	意味	モニターポートへの出力語
AS+ゴール	エージェントの生成	<エージェント名>+"enter"
AE+エージェント名, ステータス(成功/完全失敗)	エージェントの終了通知	<エージェント名>+"completely fail" (完全失敗) <エージェント名>+"complete" (成功)
MF+エージェント名, フィールド名, ゴールスタック, スクリプト, エージェント知識ベース, 実行情報	他ノードからのエージェントの移動	<エージェント名>+"enter"
MT+行先ノード名, エージェント名, フィールド名, ゴールスタック, スクリプト, エージェント知識ベース, 実行情報	他ノードへのエージェントの移動	(移動前に) <エージェント名>+"goto "+行先ノード名> (移動完了後) <エージェント名>+"bye"
ST+エージェント名, ステータス名	エージェントのステータス通知	<エージェント名>+" "+ステータス名>

【0142】すなわち、モニターは、ノード・マネージャからエージェントに関する状況報告をモニタースクリプトの形で受け取り、エージェントの状況をユーザに提示する。なお、当然ながら、モニターとのサーバ、クライアント関係が確立されていない場合は、ノードマネージャはモニタースクリプトの出力を行わない。フローチャートに示した手順は、すべてモニターとの接続が確立していることを仮定する。すなわち、接続が確立していない場合には、以下のプロトコルに従った通信を行う必要はないことはいふまでもない。

【0143】(1-5) 知識ベース

知識ベースは、宣言的な記述方法によるデータベースである。第2実施形態においては、より具体的にはProlog言語におけるファクトの集合をさす。ここで、図14は、知識ベースの構成を示す概念的ブロック図である。この図に示すように、第2実施形態の知識ベースには、アクションベースD1と情報ベースD2の2種類がある。アクションベースD1は、プランニングにおいて必須となるアクションの集合である。情報ベースD2は、プランニングにおいて必須となるシステムの初期状態を

記憶する知識ベースであり、いずれの知識ベースも、アクションによって更新することができる。

【0144】アクションベースD1及び情報ベースD2は、それぞれ、エージェント固有の部分、フィールドに区分された部分、ノード固有の部分とを有し、それぞれ、エージェントアクションベースD1A及びエージェント情報ベースD2A、フィールドアクションベースD1F及びフィールド情報ベースD2F、ノードアクションベースD1N及びノード情報ベースD2Nと称する。なお、フィールドは、エージェントの異なった目的ごとに応じた部分であり、一つのフィールドには、共通の目的を有するエージェントが対応する。知識ベースをフィールドに区分しておくことによって、異なった複数の知識ベースを1つのノード内に構成する働きを持つと共に、一つのエージェントのプランニングに、必要な情報だけを用いることによって、プランニングを効率よく実施する効果がある。

【0145】また、一般に、知識表現において、矛盾する記述の存在は、意味論的な困難を引き起こす。フィールドによる知識ベースの区分は、フィールド毎に、意味論上の表現を統一しやすくすることにより、エージェントによるプランニングの円滑な実施とアクション定義の容易な記述を促進する効果がある。

【0146】すなわち、アクションベースと情報ベースのそれぞれについて、ノードに所属する知識ベース、フィールドに所属する知識ベース、エージェントに所属する知識ベースの3種類の知識ベースが存在する。

【0147】また、第2実施形態ではPrologの用語を用いて説明する。Prolog言語については、Leon Sterling, Ehud Shapiro 著The Art of Prolog 他、多数の解説書があるが、第2実施形態におけるProlog言語記述は、標準的な記述方法のみを用いて記述している。なお、Prolog言語では、大文字で始まる名前は変数であり、マッチするアトムatom又はアトムの組み合わせである項によって置き換えられる。

【0148】なお、以下では、所定のノードからみて他のノードに関する情報すなわちmaybe 情報を用いて、エージェントがプランニングを実施し、どのように変化するネットワークに対応して挙動を行うかについて例を用いて説明する。maybe 情報はProlog言語を用いて実装できる。すなわち、知識ベースに記憶された情報のうち、maybe という種別の情報（maybe 情報）は、一応想定されるが未確認の事実を表す。maybe はフィールド情報ベースまたはノード情報ベースに格納される。

【0149】一方、本システムにおけるノード・マネー

ジャ、すなわち図1におけるエージェント管理部は、異なったフィールドに対応する様々な種類のエージェントを扱うように構成されており、例えば、エージェントには、その生成時において、エージェントの役割すなわち種類に応じたフィールドが指定される。そして、第2実施形態では、フィールド毎に知識ベースを区分することによって、プランニングで用いる知識の総量を少なくすることが可能であり、結果としてプランニングの効率を高める効果がある。

【0150】（1-5-1）アクションベース

アクションベースは、エージェントが取りうるアクションをその事前条件と事後条件とを合わせて定義したアクション定義の集合であり、アクションの集合には、ノード間でのエージェントの移動を実現する移動命令として、goアクションが含まれる。実行器がコマンドとしてgoアクションを実行した場合には、エージェントはネットワーク上のノード間で移動し、移動の前後で一貫したプランの実行を行う。

【0151】そして、アクションベース中において、一つのアクションに関する情報は、アクションの内容に加えて、アクション定義名、事前条件、事後条件及びから構成される。このうち、まず、アクション定義名は、アクションの名前と引数（アクションのパラメータ）を、項により表現したものである。また、事前条件は、アクションが実行可能となるための条件を、項のリストにより表現したものである。また、事後条件は、アクションを実行した結果の状態変化を、やはり項のリストにより表現したものである。

【0152】アクションに関する情報は、例えば、以下に示す形式によって表現できる。第2実施形態では、Prologの述語actionを用いて、この記述がアクション定義であることを示している。

action(〈アクション定義名〉, 〈アクション〉, 〈事前条件〉, 〈事後条件〉)。

ここで、〈アクション〉は、実行器によって実行可能なコマンドによって構成される。すなわち、以下に示すように複数のコマンドの列によって構成することができる。

【0153】（1-5-2）アクションベースにおける記述形式の例

ここで、第2実施形態におけるPrologを用いたアクションベースの記述形式の例を、ノードのアクションベースを例として示す。エージェントアクションベース、およびフィールドアクションベースの記述形式も同様である。

```

/*****

```

```

アクション集合

```

```

action( アクション定義名, [アクションを構成するコマンド] , [事前条件]
, [事後条件] )

```

```

*****/

```

```

action(goto ,
[
goto(Node, maybe(exist(file(F), Node)) )
],
[ home(HomeNode), maybe(exist(file(F), Node)) ],
[ add(seeking_for(file(F), HomeNode, Node)) ]
).
action(goto ,
[
update_agent_base( add( myself, returning , HomeNode)),
goto_with_goal(HomeNode, return)
],
[ have(file(F)) ],
[ add( bringing(file(F), HomeNode) ) ]
).

```

【0154】なお、プランニングでは、アクションベース中に定義された各アクションから、ゴール達成に必要なアクションを抽出して、プランを構成するスクリプトすなわちアクションの列が作成される。

【0155】(1-5-3) 情報ベース

第2実施形態における情報ベースD2も、アクションベースの場合と同じく、エージェント情報ベースD2A、フィールド情報ベースD2F、ノード情報ベースD2Nの総称であり、ネットワーク上のソフトウェア資源に関する事実の記載の集合である。例えば、エージェント情報ベースD2Aは、エージェントの挙動について、どのような動作をこれまでに完了したか等、主にその履歴に関する情報を格納する情報ベースである。また、フィールド情報ベースD2Fは、そのフィールドに固有のソフトウェア資源について、そのノードにおいてどのような処理が可能かを表現する情報ベースである。また、ノード情報ベースD2Nは、そのノードにおいて、どのような処理が可能か、どのようなリソースを有しているかに関する記述が格納されているデータベースである。もちろん、各情報ベースには、それ以外の情報を必要に応じて格納することも可能である。

【0156】なお、エージェント情報ベースD2A及びエージェントアクションベースD1Aを合わせてエージェント知識ベースDAと称する。また、フィールド情報ベースD2F及びフィールドアクションベースD1Fを合わせてフィールド知識ベースDFと称する。同様に、ノード情報ベースD2N及びノードアクションベースD1Nを合わせてノード知識ベースDNと称する。

【0157】(1-6) プランナ

プランナは、情報処理の目的として与えられたゴールに基づいてプランニングを行い、ゴールを満たすプランとしてスクリプトを出力とする部分で、エージェントの機能の一部を構成する。ここで、プランニングは、エー

ジェントによる行動プログラムの生成を指す。また、プランは、プランニングによって生成されたアクションの列であり、スクリプトは、プランを記述したソフトウェア上のデータ構造又はファイルである。スクリプトは、スクリプト・コマンドを用いて記述される。すなわち、スクリプト・コマンドは、スクリプトに記述されている実行命令である。

【0158】第2実施形態の場合、プランニングの内容は、ノードおよびフィールドおよびエージェントが固有に有する知識ベースに蓄えられた事前条件、事後条件の組（アクション定義）を用いて、与えられたゴールに到達するアクションの列を生成することとなる。なお、プランナは、必要に応じて再プランニングを行う。再プランニングは、実行が失敗した場合などに、エージェントの行動プログラムを再度生成する処理である。

【0159】プランニングは、ユーザより付与されたゴールと、アクション集合が格納されたアクションベースおよび現在の状態表現を表す情報ベースの内容を利用することによって実施可能となる。プランナは、これらの情報を入力とし、スクリプトを出力としてプランニングを実施するモジュールである。

【0160】プランナは、プランニングに際して、アクションベースと情報ベースのそれぞれについて、エージェント知識ベース、フィールド知識ベース及びノード知識ベースを読み込み、それらの内容をマージした上でプランニングを行う。

【0161】なお、ここで行われるプランニングの処理は公知のものをいれればよい（参考文献：S. C. Shapiro, Encyclopedia of Artificial Intelligenceにおけるplanningの頁を参照）。なお、プランナを実装する一例として、Prolog言語を用いて実装する場合のPrologプログラムを以下に示す。

```

/***** プランニングの定義 *****/
plan( Goal, Node ) :- plan( Goal, Node, Node, _ ), halt.

```

```

plan( Goal, Node ) :- halt.
plan( Goal, Home, Node, PL ) :-
    plan1( Goal, Home, Node, PL ),
    reverse( PL, PL2 ),
    tell( ' script' ),
    writePlanList( PL2 ), nl,
    told,
    write( ' script = "' ), write( ' script' ), write( ' "' ), nl.
plan( Goal, Home, Node, PL ) :-
    tell( ' script' ),
    writePlanList( [ ' %no' ] ), nl,
    told,
    write( ' script = "' ), write( ' script' ), write( ' "' ), nl.
plan1( Goal, Home, Node, PL ) :- info( _, _ ), !,
    infoList( Home, Info0 ),
    goal( Goal, Home, Node, Info0, Info, [], PL, [Goal], _ ),
    plan1( _, _, [ ' %no' ] ) :- !, write( ' There are no info. ' ), nl.
goal( G, _, _, Info, Info, PL, PL, AGL, AGL ) :- member( G, Info ),
goal( G, H, N1, Info0, [G|Info], PL0, [P1|PL], AGL0, AGL ) :-
    action( _, P1, Pre, [add(G)] ),
    goals( Pre, H, N2, Info0, Info, PL0, PL, AGL0, AGL ),
goals( [], _, _, Info, Info, PL, PL, AGL, AGL ),
goals( [G|GL], H, N, Info0, Info, PL0, PL, AGL0, AGL ) :-
    goal( G, H, N, Info0, Info1, PL0, PL1, AGL0, AGL1 ), !,
    goals( GL, H, N, Info1, Info, PL1, PL, AGL1, AGL ),
infoList( Node, L ) :- setof( Info, info( Node, Info ), L ).
/***** 完成したプランに含まれるアクションの各コマンドを、
スクリプトファイルとして出力 *****/
writePlanList( [] ) :- !.
writePlanList( [A|L] ) :- !, writePlan( A ), writePlanList( L ),
writePlan( [] ) :- !.
writePlan( [A|L] ) :- !, emitCommand( A ), writePlan( L ),
emitFileList( [] ).
emitFileList( [F|FL] ) :- emitFileName( F ), write( ' ' ), emitFileList( FL ),
emitFileName( file( FileName, Ext ) ) :- !,
    write( FileName ), write( ' ' ), write( Ext ),
emitFileName( DB ) :- !, write( DB ),
emitCommand( comment( G ) ) :- emitCmd( comment( G ) ),
emitCommand( X ) :- emitCmd( X ), nl,
emitCmd( update_field_base( A ) ) :- !,
    write( ' %update_field_base ' ), write( ' "' ), write( A ), write( ' "' ),
emitCmd( update_agent_base( A ) ) :- !,
    write( ' %update_agent_base ' ), write( ' "' ), write( A ), write( ' "' ),
emitCmd( check( F, Node, R ) ) :- !,
    write( ' %check ' ), emitFileName( F ),
    write( ' ' ), write( Node ),
    write( ' "' ), write( R ), write( ' "' ),
emitCmd( check_with_goal( F, SG ) ) :- !,

```

```

write(' %check '), emitFileName(F),
write(' '), write(SG), write(' ').
emitCmd( check_and_redo(F, SG, G) ) :- !,
write(' %check_and_redo '), emitFileName(F),
write(' '), write(SG), write(' '), write(G), write(' ').
emitCmd( goto_with_ack(N) ) :- !, write(' %goto_with_ack '), write(N).
emitCmd( goto(N) ) :- !, write(' %goto '), write(N).
emitCmd( goto(N, at(P, NN)) ) :- !,
write(' %goto '), write(N),
write(' remove('), write(NN), write(' '), write(P), write('
'').
emitCmd( goto(N, Reason) ) :- !,
write(' %goto '), write(N),
write(' '), write(Reason), write(' ').
emitCmd( goto_with_goal(N, G) ) :- !,
write(' %goto_with_goal '), write(N),
write(' '), write(G), write(' ').
emitCmd( wait_and_goto(N) ) :- !, write(' %wait_and_goto '), write(N).
emitCmd( put(F) ) :- !, write(' %put '), emitFileName(F).
emitCmd( get(F) ) :- !, write(' %get '), emitFileName(F).
emitCmd( rename_file(F, G) ) :- !,
write(' rename '), emitFileName(F), write(' '), emitFileName(
G).
emitCmd( plan_and_exe(G, C) ) :- !, write(' %plan_and_exe '),
write(G), write(' '),
write(C), write(' ').
emitCmd( fail(M) ) :- !, write(' %fail '), write(M), write(' ').
emitCmd( comment(G) ) :- !.
emitCmd( display ) :- !, write(' %display').
emitCmd( X ) :- !, write(X).
/***** prolog述語定義 *****/
setof(X, G, L) :- setUp, G, setPush(X), fail.
setof(X, G, L) :- setPop(L).
reconsult(F) :- [F].
append( [], X, X ).
append( [A|X], Y, [A|Z] ) :- append(X, Y, Z).
reverse( [], [] ).
reverse( [A|X], Z ) :- reverse(X, Y), append(Y, [A], Z).
member( X, [X|_] ).
member( X, [_|L] ) :- member(X, L).

```

【0162】(1-7) 実行器

実行器は、前記のプランナによって作成されたプランの
スクリプトを解釈実行する部分である。すなわち、概念的
には、エージェントがプランナーを用いて生成したス
クリプトに対し、スクリプトの中に記述されるアクション
の列をコマンドの列として解釈、実行する実行器を前
記エージェントが保有し、エージェントはプランナーと
実行器の両者を制御して動作する。

【0163】実行器における実行方式および動作技術は
一般のインタプリタ言語に準じたものである。第2実施

形態における実行器は、本質的にUNIXオペレーティング
システムにおけるcshに準じており、スクリプトを入力
することによって、スクリプトの内容を行単位で実行
し、出力として実行ステータスを返す。実行器によって
解釈実行されるスクリプトコマンドとしては、次に挙げ
るようなものを例示することができる。

【0164】(1-7-1) csh と共通の働きをもつコ
マンドの例

まず、csh と共通の働きをもつコマンドの例として
は、以下のものが存在する。これらのコマンドは、例え

ばUNIXシステム上であれば、csh を呼び出すことによって処理することが可能であるが、もちろん他の言語処理

系を用いて実装することも可能である。

【表4】

コマンド	意味	記述例
rm	ファイルの削除	rm theAgent
cp	ファイル名のコピー	
cc	コンパイラの起動または リンカの起動	cc -c src1.c cc -o exe.o src1.o src2.o

【0165】(1-7-2) 制御構造、変数の処理
第2実施形態におけるスクリプトの実行は、原則として、スクリプトの先頭から末尾に向かって順に1行に1コマンドずつ実行を行うが、次の制御構文を用いて、条

件分岐や繰り返し処理を実施することができる。動作のセマンティクスや式の記述方法の詳細は、C言語仕様のサブセットである。

【表5】

コマンド	書式	記述例
代入	\$ (変数) = 式	\$i=i+1
%if	%if (式) <任意のコマンド例> %elseif %endif	%if (i==0) rm tmpFile %elseif cp tmpFile newFile %endif
	ファイル名のコピー	
cc	c コンパイラの起動または リンカの起動	cc -c src1.c cc -o exe.o src1.o src2.o

【0166】(1-7-3) エージェント・コマンド
実行器は、モバイル・エージェントとして必要な処理を実現するために、次のようなコマンドをスクリプトの内部で使うことができる。なお、第2実施形態におい

ては、実行器を拡張することによって、コマンドの種類を拡張することができる。

【表6】

コマンド	書式	意味
goto	%goto <ノード名>	他のノードへの移動。移動に失敗した場合は、現在のゴールによる再プランを実施する。
goto	%goto <ノード名> “移動の根拠となった情報ベースの情報”	他のノードへの移動。移動に失敗した場合は、引数の情報を移動元の情報ベース(エージェント、フィールド、ノードのいずれからでも)削除する。
goto	%goto_with_subgoal <ノード名> “サブゴール”	他のノードへの移動。移動に失敗した場合は、代理手段として引数のサブゴールによるプランニングを実施
wait_and_goto	%wait_and_goto <ノード名>	一定時間を経過してから、他のノードへ移動。移動に失敗した場合も、定められた回数を超えて、再度移動を試みる。
get	%get <ファイル名>	エージェントによるファイルの獲得(コピー)
put	%put <ファイル名>	エージェントによるノードへのファイルの付与(コピー)
update_node_base	%update_node_base “更新述語”	ノード情報ベースの更新
update_field_base	%update_field_base “更新述語”	フィールド情報ベースの更新
update_agent_base	%update_agent_base “更新述語”	エージェント情報ベースの更新
check	%check <ファイル名> <根拠の出所> “根拠となった情報”	ファイルの有無のチェック。ファイルが存在しない場合は、根拠の出所から、根拠となった情報を削除するエージェントを新しく生成する。
check_with_subgoal	%check_with_subgoal <ファイル名> “サブゴール”	ファイルの有無のチェック。ファイルが存在しない場合は、スクリプト実行は失敗し、サブゴールによるプランニングを実施。
plan_and_exe	%plan_and_exe “サブゴール”	サブゴールをゴールスタックにつみ、プランニングを実施。

【0167】なお、表6最下欄における「サブゴールをゴールスタックにつみ」の意味は、図11で示した通

り、サブゴールの他に、現在のスクリプト、スクリプトの実行位置、スクリプトで用いている変数の値を、一ま

とまりのゴール・スクリプト・セットとしてスタックに積むことを意味している。このうち、goto_with_subgoal コマンドおよびcheck_with_subgoalコマンド、およびplan_and_exeは、サブゴールつきコマンドである。サブゴールつきコマンドは、ノード間でエージェントを移動させる移動命令の一種で、移動失敗時に代替手段として実行すべきサブゴールがあらかじめ指定されている移動命令である。

【0168】一方、表6の上から2番目の、「移動の根拠となった情報ベースの情報」を伴うgoto命令は、根拠付きのコマンドである。一般に、特定のゴールが与えられたエージェントがプランニングを実施した際、プランニングにおいて採用されるアクションすなわちコマンドは、その事前条件が、各種知識ベースにおいて満たされていることが必要となる。その事前条件は、maybe 情報であってもよい。ただし、maybe 情報は、実際のネットワークにおいて真である保証はないため、エージェントがプランニングを終了した後、できあがったスクリプトを実行する時点において、前提であるmaybe 情報の記述内容が異なっている場合があり得る。根拠付きコマンドとは、実行時に判明したmaybe 情報の誤りを訂正する働きを含むコマンドである。すなわち、「移動の根拠とな

った情報ベースの情報」を伴うgoto命令とは、その移動が実行できなかった場合に、プランニング時点における事前条件が異なっていたものと判定し、表6に示す通り引数に与えられた情報ベースの情報を削除する働きを持つコマンドである。

【0169】同様に、表6におけるcheck コマンドも根拠付きコマンドである。check コマンドは、それを実行するエージェントが存在するノードにおいて、その第一引数に示すファイルが存在しなかった場合は、第三引数に示す情報が誤っていたものとみなし、第二引数に示す情報ベースより削除する動作を行う。

【0170】これらの根拠付きコマンドの作用は、このコマンドを実施したエージェントによる再プランニングや、後の動作例に示す通り、後続する別のエージェントによる再プランニングの際に有効となる。

【0171】(1-7-4) 情報ベースの更新述語
表6におけるupdateコマンドは、更新器16(図7)へのインタフェースの役割を果たし、第2実施形態におけるエージェント知識ベース、フィールド知識ベース、ノード知識ベースの内容の更新を行う。更新述語には、次の表の通り2種類がある。

【表7】

コマンド	書式	意味
add	add(<<更新対象の知識ベース>>, <ファクト>)	対象知識ベースへのファクトの追加
remove	remove(<<更新対象の知識ベース>>, <ファクト>)	対象知識ベースからのファクトの削除

【0172】更新述語は、エージェントが挙動することによって新しく得られた知見を元に、情報ベースの内容を更新するための指示語である。更新述語が存在することによって、変化するネットワークにおける情報の更新を、エージェント自身が実施することによって、そのエージェント自身の再プランニングや、後続する他のエージェントによる類似したゴールに対するプランニングを、新しい情報に基づいて実施することが可能になる。

【0173】(1-7-5) 簡単なスクリプトの例
ここで、以上のようなスクリプト・コマンドを用いて、スクリプトの簡単な例を示す。このスクリプトは、ノードnode1の所定のディレクトリに存在するファイルfile1を、ノードnode0に移動し、コピーするスクリプトである。仮にプランナがこのスクリプトを出力したものとすれば、エージェントは、このスクリプトを上から下へ順に実行する。

```
%goto node1
%get file(file1)
%goto node0
%put file(file1)
```

【0174】(2) 作用

上記のような構成を有する第2実施形態は、次のような作用を有する。まず、第2実施形態におけるエージェントの活動は、ゴールを与えることによって開始される

が、以下の例では、ゴールの入力に先立って、各データベースに、次のような情報が入力されているものとす

【0175】(2-1) アクション定義の例
まず、この例で用いられるフィールドのアクションベースの定義について説明する。以下の例は、ネットワーク上に分散するファイルを収集するエージェントと、この収集のためのフィールドについて説明するものである。このような目的を達成するための5つのアクションを、以下に示すフィールドにおいて定義する。1つのアクション定義は、先に示したとおり、アクション定義名、アクション、事前条件、事後条件の組である。

【0176】アクション定義名は、便宜的につけた名称である。アクション定義における「アクション」とは、実行器のコマンドと同様、アクションによる処理の実際の内容を表す。すなわち、アクション定義における「アクション」は、そのアクション定義がプランニングによって採用された場合、プランナーが生成するスクリプトに含まれるコマンドで、エージェントが実行フェーズに移ったのち、やがて実行されるものである。

【0177】事前条件とは、アクションを実行するために必要な条件である。この条件は、プランナーがプランニングにおける処理において使用するものであり、エージェントの実行フェーズとは直接的にはかわりのない

ものである。

【0178】事後条件とは、アクションの実行の結果として追加される条件である。この条件は、プランナーがプランニングにおける処理において使用するものであり、エージェントの実行フェーズとは直接的にはかかわりのないものである。すなわち、プランニングとは実行に先立つシミュレーションである。

【0179】アクションベースには、フィールド毎に、その問題領域に応じたアクション定義が事前になされている必要がある。ここでは、アクションベースに、以下に示す第一のアクションから第五のアクションまでの5

```

action (gotol ,
      [
        update_agent_base (home (HomeNode)),
        goto (Node, maybe (exist (file (F), Node)))
      ],
      [ maybe (exist (file (F), Node)), home (HomeNode) ],
      [ add (seeking_for (file (F), HomeNode, Node)) ]
    ).

```

第一のアクション定義の内容として、具体的な2つのコマンドが存在する。コマンドupdate_agent_base は、エージェント情報ベースに対する更新操作であり、このコマンドが実行された場合、エージェントの情報ベースに、ファクト

info (home (HomeNode)).

が追加される。この操作は、このエージェントが出発したノードを、エージェントに記憶させることを意味する。本アクション定義における2番目のアクションは、根拠つきgotoアクションである。本アクションがプランニングにおいて採用された場合、生成したスクリプトを実行するエージェントは、先に実行器コマンドとして示した根拠つきgotoコマンドを実行する。根拠つきgotoアクションは、移動命令であって、当該命令の事前条件がパラメータとして添付されたものである。

【0181】そして、第一のアクション定義の事前条件は2つある。第一の事前条件は、他のノードにおけるファイルの存在に関するmaybe 述語が、情報ベース、すなわちノード、フィールド、エージェントの3つの情報ベースのいずれかに存在することである。第二の事前条件は、Prolog変数HomeNodeを束縛し、事後条件の引数に反

```

action (goto2 ,
      [
        update_agent_base ( add ( myself, returning , HomeNode)),
        goto_with_goal (HomeNode, return)
      ],
      [ have (file (F)) ],
      [ add ( bringing (file (F), HomeNode) ) ]
    ).

```

第二のアクション定義のアクションは、サブゴールつき移動コマンドgoto_with_goalである。本コマンドがプラン

ニングのアクションが定義されているものとする。すなわち、以下に説明するアクションベースは、事前にnode0, node1, node2 の各ノードに配布されているものである。なお、各ノードに対するアクションベースの配布や訂正を、第2実施形態によるエージェントを用いて行うことも可能である。

【0180】(2-1-1) アクション: gotol

第一のアクション定義では、他のノードNodeにファイルFを獲得するための移動アクションを定義している。第一のアクション定義の内容を次に示す。

映するためのものであり、エージェントの移動の状態を事後条件に正確に反映するために必要な項目である。

【0182】第一のアクション定義の事後条件は、このエージェントの状態がファイルFをネットワークにまたがって探索するために、HomeNodeからNodeに移動する状態にあることをプランナに対して示すものである。なお、このアクション定義の事後条件におけるadd は、実行器の各updateコマンドの引数に与えられる更新述語のadd (表7)とは全く異なるものである。すなわち、本記述におけるadd は、本システムのアクション設計者が、プランニングに必要なプラン途上の状態を、プランナに対して指示するものであるのに対し、実行器のupdateコマンドのadd は、スクリプトが、実行器に対し、情報ベースに対する情報の追加を指示するものである。事後条件におけるadd 述語の意味については、以下のアクションについても同様である。

【0183】(2-1-2) アクション: goto2

第二のアクション定義は、エージェントが獲得したファイルFを元のノードHomeNodeにコピーするアクションを定義している。第二のアクション定義の内容を次に示す。

ニングにおいて採用された場合、その生成されたスクリプトにおいて、先に示した実行器のgoto_with_goalコ

マンドが記述される。なお第二のアクションではgoto_with_goalコマンドに先だち、エージェント情報ベースの更新操作コマンドupdate_agent_base を行うが、これは、エージェントに対し、そのエージェントが帰還状態にあることを、明示的にエージェントに通知する働きを持つ。このコマンドが実行された場合、エージェントの情報ベースに、ファクト

```
info(myself, returning ).
```

が追加される。このupdate_agent_base とgoto_with_goalの2つの実行器コマンド操作は、エージェントが、現在存在するノードからHomeNodeへ帰還する操作がネットワーク上の障害等により一時的に失敗した場合でも、returning というサブゴールによって、再プランする機会を与える働きをもつ。

【0184】第二のアクション定義の事前条件は、エージェントが目的のファイルFをすでに獲得していること、すなわちhave (file (F)) である。第二のアクション定義の事後条件bringing (file (F), HomeNode) は、このアクションを採用した場合、エージェントの状態が、当座の目標であるファイルFを獲得し、HomeNodeに帰還する状態にあることをプランナに対して示すものである。

【0185】(2-1-3) gotoアクション

第三のアクション定義は、何らかのネットワークの障害のためにエージェントの移動が実施できなかった場合に、一つの対処方法として一定時間の間隔を開けて、移動を所定の回数試みる様なコマンドを定義している。第三のアクション定義の内容を次に示す。

```
action(wait_and_goto ,
```

```
[
wait_and_goto(HomeNode)
    action( get ,
    [
    check(file(F), HomeNode, maybe(exist( file(F), Node))),
    get(file(F))
    ],
    [ seeking_for(file(F), HomeNode, Node), maybe(exist(file(F), Node)) ],
    [ add( have(file(F))) ]
    ).
```

第四のアクション定義のアクションには、2つのコマンドが記述されている。check コマンドが、実際に実行されると、エージェントが存在するノードにおいて、ファイルFが実際に存在するかどうかを調べ、もしファイルFが存在しなかった場合には、エージェントのスクリプト実行は失敗する。check コマンドの第一引数HomeNodeは、この確認を行う根拠となった知識ベースの存在する個所を示し、第二引数maybe(exist(file(F), Node))は、根拠となった知識を示す。

【0190】第四のアクション定義の事前条件に示されている述語returning は、エージェントが所定の移動の目的を達成し、元のノードすなわちHomeNodeに帰還する

```
],
[ home(HomeNode), returning ],
[ add(return) ]
).
```

第三のアクション定義のアクションは、一定時間の間隔を開けて、移動を所定の回数試みるwait_and_go である。本アクションがプランニングにおいて採用された場合、その生成されたスクリプトにおいて、先に示した実行器のgoto_with_goalコマンドが記述される。

【0186】第三のアクション定義の事前条件に示されている述語returning は、エージェントが所定の移動の目的を達成し、元のノードすなわちHomeNodeに帰還する状態にあることを意味している。その前に記述されている述語home (HomeNode) は、変数HomeNodeを束縛する役割を果たしている。

【0187】第三のアクション定義の事後条件に示されているreturnは、このアクションを採用した場合、エージェントの状態が、HomeNodeに帰還する状態にあることをプランナに対して示すものである。

【0188】(2-1-4) get アクション

第四のアクション定義は、対象（ファイルやデータなど）を獲得する命令（例えば、put オペレーションとget オペレーション）を含み、前記エージェントは、獲得した対象を移動の際に運搬することによって、当該エージェントが生成されたノードに持ち帰る。

【0189】すなわち、このアクションはエージェント固有の状態に関するもので、エージェントによるファイルの獲得に関するものである。このアクション定義の内容を次に示す。

状態にあることを意味している。その前に記述されている述語home (HomeNode) は、変数HomeNodeを束縛する役割を果たしている。

【0191】第四のアクション定義の事後条件に示されているhave (file (F)) は、このアクションを採用した場合、エージェントの状態が、HomeNodeに帰還する状態にあることをプランナに対して示すものである。

【0192】(2-1-5) put アクション

第五のアクション定義は、エージェントによるファイルのコピーに関するものである。第五のアクション定義の内容を次に示す。

```

action( put ,
  [
    put (file (F)),
    update_field_base ( add ( HomeNode, exist (file (F), HomeNode)) )
  ],
  [ bringing (file (F), HomeNode) ],
  [ add (exist (file (F), HomeNode)) ]
).

```

第五のアクション定義のアクションには、2つのコマンドが記述されている。put アクションは、エージェントの所有しているファイルFを、エージェントが現在いるノードにコピーする。update_field_base コマンドは、エージェントが新しくファイルのコピーを行ったため、そのファイルの存在に関する情報を、エージェントが現在いるノードにおけるフィールド情報ベースに対して追加することを示している。

【0193】第五のアクション定義の事前条件に示されている述語bringing (file (F), HomeNode) は、エージェントがHomeNodeに対してファイルFを運んでいることを示している。第五のアクション定義の事後条件に示されているadd (exist (file (F), HomeNode)) は、エージェントによるコピーが終了した事後条件として、HomeNodeにおいてファイルFが存在することをプランナに対して示すものである。

【0194】(2-2) 情報ベースの記述例
次に、第2実施形態における情報ベースの記述例を、ノードの情報ベースを例として示す。
info (node0, maybe (exist (file (file1), node1))).
info (node0, home (node0)).
本情報ベースは、ノードnode0 における情報ベースの記述例である。この情報ベースにおける第一行の記述は、ノードnode0 において、ファイルfile1 がノードnode1 に存在するとする知識である。ただし、ノードnode0 とノードnode1 は異なるホストであり、この種の他のノードに関する情報は、情報の更新が日々行われるネットワークにおいて常に正しいとは限らない。maybe 情報とは、このような他のノードに関する情報である。第2行の記述は、ノードnode0 がまさしくノードnode0 であることを表現するために、第2実施形態において導入している。この例において想定するゴールは、ノードnode0 に、ファイルfile1 を獲得することである。ファイルfi

le1 は、node1 もしくはnode2 を利用する特定の人物によって作成されるファイルであることが事前に判明しているものとすれば、ファイルfile1 が存在するノードは、node1 か、node2 のいずれかであるものと仮定することができる。この事実を、ファイルfile1 の集積地の役割を果たすノードnode0 のフィールドmakeにおいて、次のような情報ベースの記述内容として表現する。この情報ベースは、第1実施形態でも述べたように、同じネットワークに属する別のフィールドのエージェント、又は別のアプリケーション・ソフトウェアを用いて、自動的に情報ベースに付加する手段を取ることが可能である。

【0195】この例では、簡単のために、ノードの情報ベースおよび、ノードのアクションベースは、空であるものと想定する。この場合、node0 におけるフィールドmakeのフィールド情報ベースは次の通りとなる。

```

info (node0, home (node0)).
info (node0, exist (file (file2), node0)).
info (node0, maybe (exist (file (file1), node1))).
info (node0, maybe (exist (file (file1), node2))).

```

ここで、述語infoは、第2実施形態において、情報ベース記述であることを明示して表現する述語であり、2つの引数を持つ。第2実施形態における述語infoは、頭部だけによって表現されるいわゆるPrologの事実 (fact) である。述語infoの頭部 (head) の第一の引数は、どのノードにおける情報であることを明示したものであり、本フィールド情報ベースの記述例においては、その内容はすべてnode0 である。info述語の頭部の第2引数は、実質的意味をもつ情報であり、Prologの項 (term) を用いて表現されている。本記述例は、次のような具体的な意味を持つ。

【表8】

home (node0)	本フィールドのホームノードが node0 である。エージェントの動作により、ファイルは node0 に集められる。
exist (file (file2), node0)	node0 に、ファイル file2 が存在する。
maybe (exist (file (file1), node1))	node1 に、ファイル file1 が存在すると想定する。

【0196】情報exist (file (file2), node0) と情報maybe (exist (file (file1), node1)) の違いは、前者においては、この記述例の情報ベースが所属するnode0 に存在す

るファイルに関する情報であり、事実として情報ベースに記載することが可能な種類の情報であるのに対し、後者のmaybe 述語は、node0 から見て他のノードであるno

del に関する事実の記載であり、変化を想定するネットワークにおいて、node0 においては記述の正当性を常に保証できない情報であることを反映したものである。

【0197】したがって、このフィールド情報ベースの4つのinfo述語の意味は、次の通りである。

- ・ このフィールドの所属するノードはnode0 である。
- ・ ファイルfile2 は、node0 に存在する。
- ・ ファイルfile1 は、node1 に存在すると想定する。
- ・ ファイルfile2 は、node2 に存在すると想定する。

【0198】次に、node1 におけるフィールド情報ベースには、次の内容が格納されてものとする。これは、file1 に関しては、その存在がnode1 か又はnode2 に存在するという、あらかじめ得られた知見に基づき、maybe の述語とするものである。

```
info(node0, maybe(exist(file(file1), node2))).
```

同様に、node2 におけるフィールド情報ベースには、次のmaybe 情報が格納されているものとする。

```
info(node0, maybe(exist(file(file1), node1))).
```

ここで説明したmaybe 情報も、先に示したinfo述語の一種である。各情報ベース、すなわちエージェント情報ベース、フィールド情報ベース、ノード情報ベースに含まれるinfo述語の一部を構成する。以上の情報は、エージェントがプランニングフェーズに移った際に、プランナに対して与えられる。これらのinfo述語は、プランナがアクションを選択する際に必要な事前条件である。

【0199】(2-3) ゴールの入力とエージェントの生成

以上のような情報を保有する第2実施形態の情報処理装置において、ノードにゴールが与えられると、ノードマネージャ15がエージェントプロセス17を生成し、プランナ13がプラン生成を行う。なお、第2実施形態において、プランナ13は、ゴールを与えることによって起動する。第2実施形態におけるPrologを用いた前記プランナの実装では、ゴール記述をPrologのクエリー（質問）として与えることによって、エージェント名をファイル名とするスクリプトを生成する。第2実施形態におけるゴールは、前述の各ベースの内容を前提する場合、

(例) `exist(file(file1), node0)`

と記述できる。このゴールは、ノードnode0 にファイルfile1 を獲得することであり、換言すれば、現状ではノードnode0 には存在していないファイルfile1 を、いずれかの他のノードよりコピーすることによりnode0 に存在させることを意味する。

【0200】このゴールをプランナに投入することの具体的な実施例としては、前記のプランナに対し、次のようなクエリーを発行することである。このゴールは、先に示したアクションベース、および情報ベースと組み合わせることによって有効となる。

【0201】

(例) `:-plan(exist(file(file1), node0), _).`

エージェント生成は、このゴール記述をnode0 のフィールドmakeのノードマネージャ15に対して与えることによって実行される。この操作は、所定の入出力手段を通じて行われる。

【0202】ここで、エージェントの生成を含むノードマネージャの動作手順を図15に示す。ノードマネージャは、当該ノードに存在するエージェント（アクティブ・エージェント）を登録するリスト（アクティブ・エージェント・リスト）ALを有する（図13）。そして、ノードマネージャ15は、図15に示すように、他のノードマネージャやエージェントプロセスからの指示として、ノード・ポートNP（図13）から通信内容を受け取り（ステップ1501～1505）、指示として与えられる文字列stringの内容に応じて（ステップ1506～1510）、エージェントの生成（ステップ1511, 1512）、消滅（ステップ1521, 1522）又は移動など（ステップ1531～1553）の処理を行い、アクティブ・エージェント・リストを更新する。

【0203】なお、ノードマネージャは、このように送られる語の内容を解釈することによって各々のインタフェースの処理内容を決定するが、語をどのように転送するかには、さまざまな方式がありうるので、所望の方式を自由に採用してよい。

【0204】エージェントマネージャ15によって生成されたエージェント・プロセス17（図7）は、ノードマネージャ15によって生成／管理されながら、全体として、エージェントAの動作を実現する。なお、ノードマネージャ15は、エージェント・プロセス17を同時に複数生成することができる。

【0205】エージェントの生成後、ノード・マネージャ15によってエージェント名がそのエージェントに与えられる。エージェントはネットワーク内のノード間で移動するので、エージェントを識別するためには、エージェント名は、ネットワーク内における唯一の名称である必要がある。ノード・マネージャ15は、そのノード名、時刻、およびそれまでに生成したエージェントの数等の情報を組み合わせて、ネットワークにおけるユニークなエージェント名を、生成したエージェントに与える。生成されたエージェントは、以下に説明するように、プランニング、実行、移動の3つのフェーズを有し、必要に応じてそれぞれのフェーズに入る。

【0206】(2-4) プランの作成

ゴールが与えられてエージェントが生成されると、プランナ13が、与えられたゴールに対するプランニングを実施する。すなわち、最初にゴール

```
exist(node0, file(file1)).
```

が所定の入出力装置から与えられることによってエージェントが生成され、エージェントは、プランニングフェーズに入る。すなわち、エージェントはゴール・スクリ

プトセットをゴールスタックに積み、プランナ13を起動することによってプランニングを実施する。

【0207】図16は、プラン作成の内容をフローチャートの形式で示した概念図で、公知であるプランニング技術を説明した図である。すなわち、第2実施形態におけるプランナ13は、ゴールより後ろ向き推論を実施し、事前条件と事後条件を参照しながら、ゴールを満たすアクション系列を求める。なお、図16は、言い換えれば、実際のプランニングの動作を簡素化して記述したものであり、実際のPrologの動作において発生する変数とアトムの一化(unification)等の動作を省略して簡素化して記述したものである。

【0208】このプランニングでは、第1実施形態と同様に、ゴールを事後条件とするアクション定義を探索する。次に、見つかったアクション定義中の事前条件をそれぞれゴールとして、そのゴールを事後条件とするアクション定義を探索する。してプランのスクリプトに書き出し、書き出したアクションをゴールとして、さらに同様の手順を繰り返すことによってプランを作成する。すなわち、具体的には、ゴールから(ステップ1601)、アクションput, goto2, get, goto1を辿って全ての条件が満たされるまで(ステップ1602~1605)、手順を繰り返す。この探索は、ゴールから現在の状態に向かって逆向きに行っているため、選択されたアクション定義を選択した順序とは逆向きにして、それらのアクション定義中のアクションの内容、すなわちコマンドの並びを、スクリプトとして生成する。なお、このとき、どのようなアクション定義の順序をとっても全ての条件を満たすことができなかった場合が、プランニング失敗である。

【0209】図15の処理によって、node0のフィールドmakeのフィールド知識ベースおよびnode0のアクションベースに基づき、次のようなプラン結果がスクリプトとして得られる。左端の数字は、行番号である。

```
1 %update_agent_base "add(myself, home(node0))"
2 %goto node1 "maybe(exist(file(file1), node1))"
3 %check file(file1) node0 "maybe(exist(file(file1), node1))"
4 %get file(file1)
5 %update_agent_base "add(myself, returning, node0)"
6 %goto_with_goal node0 "return"
7 %put file(file1)
8 %update_field_base "add(node0, exist(file(file1), node0))"
```

【0210】なお、プランナ13は、表6のコマンドgoto(2番目)に示すように、プランの一部としてエージェントをノード間で移動させる移動命令を作成する際に、当該命令の前提となる事前条件を添付する(請求項11)。すなわち、node1への移動の根拠が

"maybe(exist(file(file1), node1))"

であることを意味している。なお、事前条件を移動命令に添付する場合、事前条件の全部又は一部を移動命令の付属パラメータとする。

【0211】このように、第2実施形態では、移動命令に事前条件が(根拠として)添付される。そして、当該移動命令に基づく移動が失敗する場合は、命令が前提とした事前条件が成立していなかった可能性が最も大きい。この事前条件は、命令に添付されているので、命令を参照すれば容易に判明する。このため、移動失敗への対応が容易になる。

【0212】なお、プランナ13は、ノードのアクションベースおよび情報ベース、フィールドのアクションベースおよび情報ベース、エージェントのアクションベースおよび情報ベースを、プランニングの実施に先立って読み込む。第2実施形態では、以上を総称して知識ベースと呼ぶ。第2実施形態におけるPrologを用いた実装においては、既に示したように、各情報ベースの情報をPrologの節により表現しているため、プランニングに先立ち、以上のすべての知識ベースをPrologのアサートを用いて、読み込むことにより、プランニングを実施することができる。

【0213】前記のクエリーにより、ファイル名'script'というスクリプトが一度生成される。そして、エージェント・プロセスは、その該当する固有のエージェント名によってファイル名をつけかえ、他の同時に存在するエージェントのスクリプトと区別する。もちろん、複数のエージェントが同時にプランニングを実施した場合は、両者は平行プロセスとなるため、ファイルの生成、削除において、エージェントの動作の同期が必要となるが、適切なロック機構を導入することによって、複数のエージェントが同一のノードに存在することが可能である。

【0214】プランニングは、ゴールの内容、ノードの知識ベース、フィールドの知識ベース、移動の知識ベースの内容によって、成功する場合と失敗する場合とがある。成功の場合は、プランニングの出力であるスクリプトが生成され、エージェントは実行フェーズに移る。なお、エージェントは生成時に与えられたゴール以外に、途中での失敗を回復するなどのためのサブゴールを複数格納できるゴールスタックを有している。そして、サブゴールのプランニングに失敗した場合は、より上位のゴールによってプランニングを実施する。最終的にすべてのプランニングに失敗した状態が完全失敗である。完全失敗の場合は、エージェントは終了処理される。

【0215】なお、プランニングの結果、前掲のスクリプトを得るので、エージェントは、これをゴールスタック最上段のゴール・スクリプトセットに格納する。次に、エージェントは実行フェーズに移る。

【0216】プランニングでは、最終的な目的であるゴ

ールを達成するための中間的なゴールを作成し、この中間的なゴールを達成するための子エージェントを生成するようにしてもよい（請求項12）。このようにすれば、中間的なゴールの作成が子エージェントに依託されるので、エージェント毎の情報処理の内容が単純化されることによって、情報処理が効率化される。また、複数のエージェントが並行または並列動作することによって、情報処理が迅速化される。

【0217】（2-5）プランの実行

スクリプト記述を生成すると、エージェントは実行器16を用いて実行フェーズに入る。後述するように、スクリプトは、制御構造を有するコマンドの列である。実行フェーズにおけるエージェントは、スクリプトを制御構造にしたがって解釈し、順にコマンドを実行する。

【0218】図17は、プランを実行する手順を示すフローチャートである。この手順では、スクリプトが終了するまで実行行番号を増大させながら1行ずつコマンドを読み出し（ステップ1701～1704、1715、1716）、移動命令ならばその旨をエージェントマネージャに依頼し、移動命令以外はコマンドの種類に応じた処理を行う（ステップ1708）。なお、実行途上でエラーが発生した場合は（ステップ1709）、エラーの生じたコマンドの種類毎に所定のエラー処理を行う（ステップ1710～1714）。

【0219】すなわち、実行器16は、スクリプトを読み込み、一行単位でコマンドを切り出し、コマンド名およびスクリプトを解釈し、コマンドの種類に応じて、先に示した表に示したコマンドを実行する。実行器が実行するスクリプトの文法には、さまざまなバリエーションが存在し得るが、どのような文法を解釈する実行器を使用するかに関しては、本発明の本質とはかかわりなく、表に示すような移動コマンドおよび表に示す情報ベースの更新を伴うコマンドや、根拠となった情報を明示するコマンドが実行器に用意されている点に特徴がある。これらのコマンドが用意されていることにより、エージェントに適切なサブゴールを与えたり、あるいはネットワークにおける変化に伴って適切に情報ベースの更新を行うことが可能になる。

【0220】コマンドの実行が失敗した場合、又は再プランコマンドが実施された場合には、所定のゴールに基づいてプランニングを行うためのプランニングフェーズに戻る。移動コマンドが実施された場合は、移動フェーズに移る。

【0221】（2-6）エージェントの移動

実行器16が、実行中のスクリプトにおいて移動命令に遭遇すると、その旨がノードマネージャ15に通知され、エージェントが他のノードに転送される。エージェントを移動するために実際に転送される情報は、スクリプト、エージェント、ゴールスタック、エージェント知識ベースである。これらの情報を、すべて移動先のノードに転送することに成功した場合、エージェントは、移動先において実行フェーズに戻る。

【0222】（2-6-1）他のノードへのエージェントの移動

図18は、他のノードへのエージェントの移動する場合の移動元側ノードのノードマネージャの動作を示すフローチャートである。すなわち、エージェントに関する必要な情報を収集すると共に（ステップ1802～1804）、移動先のノードをホストとして通信を試み（ステップ1805、1808）、通信路の確立に失敗した場合は移動をあきらめ（ステップ1806、1809）、エージェントプロセスに対して失敗を通知する（ステップ1807、1810）。通信路が確立できた場合は、移動先に必要な情報を送信して返答を待ち（ステップ1811、1812、1815）、通信路が確立できた場合でも、相手先ノードからの返信を受け取ることができなかった場合は、同様に移動失敗とみなし（ステップ1813）、エージェントプロセスに対して通知を行う（ステップ1814）。

【0223】（2-6-2）他のノードからのエージェントの移動

図18の処理に対応して、移動先のノードのノードマネージャは、図19に示す処理によってエージェントを受け入れる。すなわち、ノード・マネージャは、他のノードからノード・ポートにプロトコルMFを受信した場合、必要な情報を受け取って（ステップ1901）、新たなエージェントに資源を割り付け（ステップ1902）、新たなエージェントの情報を所定の記憶領域に保存したうえ（ステップ1903）、移動元に受け入れ成功を返答して（ステップ1904、1905）、エージェントプロセスを起動する（ステップ1906）なお、この処理の途上で、通信の異常が発生した場合は、JavaやC++のオブジェクト指向言語において用いる例外処理機構によって、処理される。

【0224】なお、図20は、エージェント＝ノード間インターフェースと、ノード＝ノード間インターフェースを用いて、エージェントがどのように移動するかを示す補足的な説明図である。エージェントは、ノードマネージャを介して、間接的に移動を行う。

【0225】（2-7）失敗への対応

なお、スクリプトの実行又は移動が失敗した場合すなわちプランの実行が失敗した場合、プランの実行を中断し、再度プランを作成して実行する。失敗が発生する態様としては、例えば、エラーや例外の発生などが考えられる。第2実施形態では、プランの実行や移動が失敗しても、再度プランが作成されるので、情報処理が円滑に継続される。

【0226】具体的には、プラン中のアクションの実行又は移動に失敗した場合、実行又は移動の失敗を回復するためのサブゴールを生成し、当該サブゴールの実行終

了後に前記プランの実行を再開するための情報を保存し、前記サブゴールに基づいてプランの作成及びプランの実行を行い、サブゴールに基づくプランの実行後に、保存した前記情報を用いて元のプランの実行を継続する。

【0227】例えば、goコマンドに基づくエージェントの移動が失敗し、かつ、当該goコマンドに根拠が付加されていない場合は、現在のスクリプトを生成したゴールを、再びゴールとして用いて再度プランニングを行う。移動が失敗する場合としては、移動先ノードへのネットワーク接続が確立されていない場合、又は一定時間以内に移動が完了しない場合、又は移動先においてメモリやディスクスペース等の必要なリソースが確保されない場合が考えられる。このような場合は、移動命令失敗とみなして、所定のサブゴールに基づくプランニングフェーズに戻る。

【0228】このような失敗は、望ましくは移動元側で検出する。例えば、エージェントがgoコマンドを実施する際に、エージェントの移動を管理するノードマネージャにおいて、移動先ノードの何らかの不具合により、エージェントが移動できない場合に、エージェントの移動失敗を移動元で検知する。

【0229】このように、失敗の際にサブゴールを作成してプラン作成を行うことは、実行フェーズにあるときでも、必要に応じてプランニングフェーズに移行することを意味する。なお、保存する情報としては、元のゴール、元のスクリプト、当該スクリプトの実行位置等が考えられる。また、これら情報を保存するには、これら情報をゴールスタックに積み重ねる。そして、作成されたサブゴールのうち、もっとも下位のサブゴールよりプランニングを実施し、作成されたプランを実行する。実行後に元のプランを再開するには、保存した前記情報をスタックより取り出して用いればよい。

【0230】また、第2実施形態では、プランニング、プランの実行又はプランに基づく移動が失敗した場合に、失敗の原因となった情報が修正される。このため、それ以降の失敗が減少し、処理が効率化される。例えば、実行器16は、事前条件がパラメータとして明示されているgoコマンドの実行が失敗した場合、当該事前条件をローカル情報すなわちフィールド情報ベースから削除する。

【0231】なお、サブゴールを作成し及び情報を修正する態様は、アクション毎若しくは移動先毎にあらかじめ定義された方式、又はプラン作成で用いられたいずれかの情報によって定義された方式に基づいて、また、ローカル情報などに基づいて定めればよい。

【0232】このように、失敗時への対処の場合を含めて、第2実施形態では、プランの実行においてサブゴールを作成し、サブゴールについてさらにプランの作成と実行を行うことによって、階層的なゴールの構造を用い

て情報を処理する。この場合、エージェントは、goアクションによって移動する際に、階層的なゴールスタックを伴って移動するので、移動先のノードでも移動前と同じゴールスタックを用いて、一貫性のある動作を行うことができる。

【0233】このような再プランニングを円滑に行うため、プランを構成するアクションの一種として再プランコマンドを用い、当該再プランコマンドは、プランの実行中に実行されるとプラン作成手段がサブプランを作成する。

【0234】なお、移動後にプランニングに失敗し、事前条件を成立させるようなアクションが最終的に生成できない場合には、スクリプトの実行を失敗終了させ、移動前のノードに戻り、スタック中に保存してあるゴールを用いて再プランニングすればよい。また、ゴールに対するプラン生成が最終的に不可能な場合や、又は実行不可能なゴールを受け取った場合は、電子メール、その他の手段により、必要に応じて発信者に通知するようにしてもよい。

【0235】エラー対策の別の態様としては、前記プランを作成する手段は、プランの前提となっている事前条件がプランの実行時に満たされているか否かを確認するためのスクリプトを、当該プランに加えることが考えられる。これによって、プラン作成時点での条件が満たされていることが実行時に確認される。すなわち、プランニングが行われた時点と、作成されたプランが実行される時点との間には、時間的な間隔が必然的に存在するため、プランニング時点において満たされていた条件が、実行時にはすでに満たされていない可能性が存在する。事前条件の確認によって、プランの適切な実行が確保される。

【0236】(2-8) 終了処理

終了処理の一例としては、エージェントが情報処理に成功した場合に、成功の旨のメールをゴールの発行者に送信し、失敗の場合は、失敗した旨のメールを送付する処理が考えられる。その他にも、終了処理として様々な実現態様が考えられるが、エージェントの生成時に、どの終了処理の実施形態をとるかに関して情報を付加しておくことによって、終了処理のバリエーションを持たせるようにしてもよい。

【0237】(2-9) エージェントの詳細な動作

以上のような各作用を、エージェントを中心に示した処理手順を図21のフローチャートに示す。すなわち、エージェントの生成時に、ノードマネージャよりゴールを受け取るとこのゴールをゴール・スタックに積む(ステップ2101)。ゴールスタックが空かどうかをチェックし(ステップ2102)、ゴールスタックが空の場合は、このエージェントが完全にゴール達成に失敗したと判定し、後述する終了時処理を実施し、終了する(ステップ2103)。

【0238】ゴールスタックが空でない場合は、ゴールスタックよりゴールをPOPし、そのゴールに対して、後述するプランニングを実施する（ステップ2104）。プランニングが失敗した場合は、スタックの内容をチェックするステップに戻る（ステップ2105）。

【0239】プランニングに成功した場合は、スクリプトが生成される（ステップ2106）。このとき、スクリプトの実行位置を更新し（ステップ2107）、プランニングフェーズから実行フェーズに移る。

【0240】プランニング及びスクリプトの生成では、新しくエージェントが作成された直後のプランニング、においては新規にスクリプトが生成される。また、プランニング及びスクリプトの生成では、新規に作成された直後のプランニングにおいては、プランニングの前の時点に存在したスクリプトに対して、プランニングした結果が追加される。プランニング及びスクリプトの生成において、プランニングに成功しなかった場合は、再び、ゴールスタックが空かどうかをチェックするステップに戻る。

【0241】プラン生成に成功すると、スクリプトの解釈実行に移る。スクリプトの解釈実行は、後述するエージェントの実行器によって処理される。スクリプトの解釈実行の動作に関しては、BASICなどの一般のインタプリタ言語と本質的に同様のものである。実行器は、スクリプトの現在の実行位置を次の位置に移動させる事により、コマンドを読み取り、コマンドの種別、およびコマンドの引数を解釈し、実行する。

【0242】実行器の解釈部において、現在の実行位置のコマンドが移動コマンド（goto）であった場合は、移動処理ステップに移る（ステップ2108）。移動処理ステップにおいて、後述する移動処理を行う。移動処理は、ノードマネージャによるノード間にまたがる処理であり、移動元からはエージェントが消え、移動先にエージェントが復元される（ステップ2109）。この移動が完了した場合は（ステップ2110）、移動先のノードにおいて図21のフローチャートの処理が継続される。

【0243】移動が失敗した場合は（ステップ2110）、移動元のノードにおいて、移動命令が、ゴールを伴う移動コマンドであるかどうかを調べる（ステップ2114）。ゴールを伴わない移動コマンドの場合は、次にプランニングフェーズに入るためにゴールスタックが空かどうかを調べるステップに戻る。ゴールを伴う移動コマンドの場合は、後述する失敗時処理を実行し（ステップ2115）、サブゴールをゴールスタックにPUSHし（ステップ2116）、再びプランニングのステップに戻る。

【0244】実行位置のコマンドが移動命令でなかった場合は（ステップ2108）、次に実行器はスクリプトの終了かどうかを調べる（ステップ2111）。スクリ

プトが終了の場合は、エージェントの成功終了と判定し、成功時の終了時処理を行った上で（ステップ2117）、終了する。

【0245】ステップ2111においてスクリプトが終了でない場合は、実行位置のコマンドを解釈し、実行する（ステップ2112）。この処理において、コマンド実行が成功した場合は、ステップの処理に戻る。また、コマンド実行が失敗した場合は、ステップ2114の処理に移る（ステップ2113）。

【0246】以上は、移動するエージェントの立場より見た挙動である。エージェントは、全体として最初に与えられたゴールを満たすために、スクリプトを生成し、スクリプトに基づいた一貫性のある挙動を示す。

【0247】（3）スクリプトの実行とノード移動の実例

続いて、前記のプランナによって生成されたスクリプトを実行する実行フェーズおよび移動フェーズにおけるエージェントの挙動について、先述のスクリプト（以下「第一のスクリプト」という）を具体例として説明する。なお、図22は、具体例において、失敗への対応を含めた処理を示すフローチャートである。

【0248】（3-1）成功例

最初に、プランニングにおける想定がすべて正しく、また実行時にエラーが発生しなかった場合の成功例について、具体的な動作を示す。

【0249】まず、先に述べたプランのうち、1行目の`%update_agent_base "add(myself, home(node0))"`の実行により、エージェントの情報ベースに情報、`info(myself, home(node0))`を追加する（ステップ2201）。この操作は、このスクリプトの実行がこの後に何らかの理由により失敗し、再プランニングを実施する場合に、自分自身の出発したノードをプランナに伝えるために必要となる。

【0250】次に、2行目の`%goto node1 "maybe(exist(file(file1), node1))"`の実行により、`node1`への根拠つき移動命令であり、この命令によりエージェントはノード1に移動する（ステップ2201）。引数の根拠の効果については、後述する。

【0251】続いて、3行目の

`%check file(file1) node0 "maybe(exist(file(file1), node1))"`の実行により、ノード`node1`において`file1`が実際に存在するかどうかを確認を行う（ステップ2205）。この例では、プランニングで想定した通り、`file1`が`node1`に存在したものとする。この場合は、`check`コマンドの第2引数、第3の引数は意味を持たない。

【0252】そこで、4行目の

`%get file(file1)`の実行により、移動した先である`node1`から`file1`をエ

エージェントの所有ファイルとする（ステップ2209）。以後、エージェントは同ファイルをput するまで、移動とともにfile1 を持ち運ぶ。

【0253】5行目の

```
%update_agent_base "add(myself, returning, node0)"
の実行により、エージェントの情報ベースに情報
info(myself, returning, node0)
```

を追加する（ステップ2209）。この操作は、この後のエージェントのnode0への帰還が何らかの理由により失敗し、再プランニングを実施する場合に、エージェントが帰還状態であることをプランニングに反映するために行う。

【0254】そして、6行目の

```
%goto_with_goal node0 "return"
```

の実行により、エージェントは、node0 に移動する（ステップ2209）。また、7行目の

```
%put file(file1)
```

の実行により、エージェントが所有するfile1 をnode0 にコピーする（ステップ2214）。最後に、8行目の

```
%update_field_base "add(node0, exist(file(file1), node0))"
```

の実行により、エージェントは、node0 のフィールド情報ベースに、info(exist(file(file1), node0)) の追加を行う（ステップ2214）。以上のように、エージェントは、スクリプトを実行する途上で実行失敗を起こさない限り、最初に与えたゴールを満たすように動作する。

【0255】（3-2）失敗例

次に、失敗例に基づいて、再プランニング実行時のエージェントの動作について説明する。node0 で行ったプランニングは、node0 において知る限りの情報を用いてプランニングしたに過ぎない。実際にはネットワークに接続される各ノードや、ネットワークの状態は、時間の経過と共に変化するために、プランニングによって生成したスクリプトの実行は、その途中で失敗する可能性がある。

【0256】エージェントの実行の失敗には、次ような理由が考えられる。

- (1) 故障、保守、過負荷のためにノード又はネットワークが一時的に稼動しない。
- (2) エージェントがプランニングに使用した情報ベースの内容が誤っている。
- (3) 情報ベースの内容が、時間と経過によって古くなった。

【0257】図16に示すように、本動作例においては、次のようなケースが考えられる。

- ・ node0からnode1 へのエージェントの移動が失敗する。
- ・ ノードnode1 には、file1 が存在せず、エージェントがファイルを獲得できない（かわりにnode2 に存在す

る）。

- ・ ファイルfile1 のnode1 からnode0 へのエージェントの移動が失敗する。

【0258】以下では、以上のケースにおいて、本発明の一実施例である情報処理装置がどのように対処するかについて説明する。

【0259】（3-2-1）第1の失敗例

最初に、スクリプトの2行目%goto node1 "maybe(exist(file(file1), node1))"において（ステップ2201）、根拠つきgoto命令のnode0 からnode1 への移動が失敗した場合について説明する（図22の失敗シナリオ1）。この場合（ステップ2202）、ノードマネージャは、図18に示した動作を行い、エージェントに対して移動不可能を通知する。この場合、エージェントの実行器は、これを受けてプランニング時点において、移動の根拠となった情報すなわち第3引数の知識をノード、フィールド、エージェントの各情報ベースより削除する（ステップ2203）。根拠の削除によって、各情報ベースは次のように変化する。

【0260】まず、node0 のノード情報ベースは、第2実施形態の場合には、もともと内容がないため、変化を受けない。また、エージェント情報ベースの場合は、それに先立つ第一のスクリプトの1行目のupdate_agent_base コマンドの実行により、info(myself, home(node0)). という節（英語clause）が1つ加えられているが、maybe 述語の節はエージェント情報ベースには存在しないため、何も変化を受けない。

【0261】一方、第1の失敗の時点におけるノードnode0 のフィールドmakeの情報ベースの内容は、

```
info(node0, home(node0)).
```

```
info(node0, exist(file(file2), node0)).
```

```
info(node0, maybe(exist(file(file1), node2))).
```

であるため、この作業の後、実行器は、エージェントプロセスに実行失敗のステータスを返し、エージェントはプランニングフェーズに移行する。失敗したgoto命令は、サブゴールを持たないコマンドであるため、図21の動作に基づき、現在のゴールスタックの内容で、再プランニングを実施する（ステップ2204）。

【0262】上記のフィールド情報ベース、およびエージェント情報ベース、前述のアクションベース、前述のプランナによる再プランニングによって、次のような第二のスクリプトが生成される。

【0263】

```
%update_agent_base "add(myself, home(node0))"
```

```
%goto node2 "maybe(exist(file(file1), node2))"
```

```
%check file(file1) node0 "maybe(exist(file(file1), node2))"
```

```
%get file(file1)
```

```
%update_agent_base "add(myself, returning, node0)"
```

```
%goto_with_goal node0 "return"
```

```
%put file(file1)
```

```
%update_field_base "add(node0, exist(file(file1), node0))"
```

このスクリプトが第一のスクリプトと異なる点は、file 1 を取得する対象がnode1 からnode2 に変更されている点である。

【0264】すなわち、実行の失敗において、情報ベースの更新が適切に行われない場合には、再プランを行っても、第一のスクリプトと同一のスクリプトを生成し、エージェントは同じ失敗を繰り返す恐れがある。第2実施形態では、根拠つきgoコマンドを用いることにより、フィールド情報ベースの内容から、file1 がnode1 に存在するとする情報

```
info(node0, maybe(exist(file(file1), node1)))
```

が削除された。このため再プラン時において、file1 が存在し得る次の候補となるnode2 を用いたプラン生成が行われた。このように、第2実施形態によれば、ソフトウェア・エージェントに求められるエージェントの自律性を向上し、ネットワーク上の変化へのエージェントの対応能力を高めることが可能となる。

【0265】(3-2-2) 第2の失敗例

次に、第一のスクリプトにおいて、node0 からnode1 への移動は成功したが、node1 には、file1 が存在しなかった場合の本動作例におけるエージェントの挙動について説明する(図22の失敗シナリオ3)。この場合は、node1 に移動したエージェント実行器が第一のスクリプトの第3行目のcheck コマンドにおいて、file1 の存在確認を行う際に、実際にはfile1 が存在しないため、実行失敗が発生する(ステップ2205)。

【0266】check コマンドは、実行器コマンドの表で示した通り、その第2引数が、根拠と出所、すなわち根拠となる情報が格納されていた情報ベースの存在するノードであり、その第3引数が、根拠となる情報である。第1の失敗例と同じく、この場合にも、その根拠となる情報の更新を実施する(ステップ2207)。

【0267】ただし、第1の失敗例の場合と第2の失敗例の場合が本質的に異なるのは、すでにエージェントはnode1 に移動している点である。したがって、情報ベースの内容を更新するためには、再び元のノードに移動しなければならない。ここでは、誤っている情報ベースが存在するノードをcheck コマンドにおいて明示する方法により、そのノードのフィールド情報ベースおよびノード情報ベースの(両者の)更新を行うという方法をとる。

【0268】このとき、情報ベースの内容を更新するための実施手段としては、ノードマネージャを介したノード間通信を用いた情報ベースの更新か、又は情報を更新するゴールを別途発行し、第二のエージェントを別途生成して、独立して動作させるという手段もある。いずれの場合でも、根拠付きコマンドの存在によって、情報ベ

ースの更新が可能となる。

【0269】node1 における再プランにおいて、使用する情報ベースは、ノードnode1 のノード情報ベースと、ノードnode1 のフィールドの情報ベースと、エージェントの情報ベースである。

【0270】第2の失敗例では、失敗後の更新における、エージェントの情報ベースの内容は、

```
info(myself, home(node0)).
```

であり、一方、node1 のノード情報ベースの内容は空であり、node1 のフィールドmakeのフィールド情報ベースの内容は、

```
info(node1, maybe(exist(file(file1), node2))).
```

である。これら2つの情報から、失敗例1と同様に、元のゴール exist(file(file1), node0)に対する再プランニングを実施すると(ステップ2208)、第1の失敗例と同様な次のスクリプトが得られる。異なる点は、第1の失敗例ではエージェントはnode0 からnode2 へ移動するのに対し、第2の失敗例ではnode1 からnode2 への移動のプランを立てたことである。

```
%update_agent_base "add(myself, home(node0))"
```

```
%goto node2 "maybe(exist(file(file1), node2))"
```

```
%check file(file1) node0 "maybe(exist(file(file1), node2))"
```

```
%get file(file1)
```

```
%update_agent_base "add(myself, returning, node0)"
```

```
%goto_with_goal node0 "return"
```

```
%put file(file1)
```

```
%update_field_base "add(node0, exist(file(file1), node0))"
```

これは、エージェント情報ベースの存在により、エージェントのホームノードの位置を他のノードに移動した際に利用できるようにしているためである。このように、移動におけるエージェントの一貫性を保持する効果がある。

【0271】(3-2-3) 第3の失敗例

次に、第3の失敗例として、第一のスクリプトにおいて、node1 にてfile1 のget までは成功したが(ステップ2209)、最後にnode0 に帰還する移動の際にエージェントの移動が失敗する場合の本動作例のエージェントの挙動について説明する。

【0272】失敗は、node1 において、第一のスクリプトの6行目

```
%goto_with_goal node0 "return"
```

にて発生する。これは、サブゴールを伴う移動命令であり、ここでは、第3の引数"return"がサブゴールである。したがって、本例題においては、同じ移動失敗であっても、エージェントによるその移動失敗への対処方法は、第1の失敗例の場合と必然的に異なっていなければならない。

【0273】エージェントの実行器は、図21に示す手

順により、サブゴール付きコマンドgoto_with_goalのnode1からnode0への移動失敗をノードマネージャより通知を受けて、実行を失敗する。この場合、その時点で実行しているスクリプトについて、現在の実行位置等の情報とともに、サブゴール“return”をスタックにつみ、再プランを行う（ステップ2211）。

【0274】再プラン時のエージェント情報ベースの内容は、goto_with_goalコマンドの実行に先立って、第一のスクリプト5行目において、内容追加が存在するため、次のようになる。追加された知識retruningは、エージェントが、ホームノードであるnode0に帰還する状態にあることを明示する効果をもつ。

```
【0275】%info(myself, home(node0)).
%info(myself, returning).
```

一方、失敗例2の場合と同じく、node1のフィールドmakeの情報ベースの内容は、次の一行である。

```
【0276】%info(node1, maybe(exist(file(file1), node2))).
ノードnode1のノード情報ベースの内容は空である。
```

【0277】これらの情報ベース、先に示したアクション定義およびプランナによって生成される第四のスクリプトは、次の1行である。

```
%wait_and_goto node0
```

このコマンドは、一定間隔を開けて移動を所定の回数試みるコマンドである（ステップ2212）。ネットワークが一時的な不具合のため、前回のnode0への移動が運悪く失敗したものと想定している。本例題の、初期のエージェントのゴールexist(file(file1), node2)は、node0の存在を暗に要求しており、node0がネットワークの変化によって、消滅した可能性まで考慮する必要はない。したがって、node1からnode0への移動失敗は、一時的な障害であると仮定することは妥当である。

【0278】再プランした結果のスクリプトであるwait_and_goto node0を実行してもnode0へ移動できない場合は（ステップ2213）、これ以上のサブゴールはないため、エージェントは図21の手順に従い、ゴールスタックをポップし、最初のゴールexec(file(file1), node1)を用いてさらに再プランする。この場合に生成されるスクリプトは、第2の失敗例の場合と同じであり、node2に移動してからnode0に戻る。さらに、このnode2へ移動するプラン実行も失敗した場合は、エージェントは完全失敗となる。所定の終了処理を実施し、エージェントが完全失敗した事をいずれかのノードにおいて記録する。

【0279】再プランした結果のスクリプトであるwait_and_goto node0を実行によってnode0への移動ができた場合は、この第四のスクリプトは、成功して終了し、エージェントは図21の手続きにしたがって、ゴールスタックをポップし、node0において第一のスクリプトの第7行目より実行を再開する。このようなサブゴール実

行は、この実施例におけるエージェントのnode0への帰還といった場合に発生した局所的な目的に対する不具合を、移動先ノードの知識を利用して代替手段を探す効果がある。

【0280】また、エージェント内のゴールスタックの存在、およびその移動における運搬は、そうした局所的な代替手段が成功しなかった場合に、より大きなゴールでプランニングをやり直すチャンスを与える効果がある。

【0281】アクション定義における第1のアクションと第2のアクションは、いずれも移動命令であったが、第一のアクションがファイルFを捜し求めるための移動であったのに対し、第2のアクションは、エージェントが移動先での所定のコマンド実行を達成し、移動元に帰還するためのアクション定義であるという違いを定義することができた。

【0282】以上のように、第2実施形態では、第1及び第3の失敗例に示したように、同じ移動命令の場合でも、エージェントの文脈に応じたサブゴールを発行するアクション記述が可能となることにより、多くの予期せぬ変化に対応できる自律的なエージェントを記述することが可能となる。

【0283】（4）第2実施形態の効果

以上のように、第2実施形態では、エージェントがプランに基づいてノード間で移動するだけでなく、移動先のノードでもさらにプラン作成が行われる。このため、プランの実行段階や移動先ノードの状態に応じて、情報処理の内容が柔軟に決定され、情報処理が効率化される（請求項8、13、14）。

【0284】また、第2実施形態では、プラン作成の際、エージェントの種類に対応する情報のみがいられるので、プラン作成が効率化される。また、ノード間での移動の際も、エージェントは対応するフィールドに移動するので、移動前と同様に対応するフィールドの情報をを用いてプランの実行やプランの作成が行われる（請求項9、15）。

【0285】また、第2実施形態では、エージェントが固有の情報と共に移動し、移動先のノードでも固有の情報を用いるので、個々のエージェントの状態やアクションに適したプランニングが可能となる（請求項10）。

【0286】3. 他の実施形態

なお、本発明は、上記各実施形態に限定されるものではないので、次に例示するような他の実施形態をも包含するものである。例えば、各ノードは相互に独立したハードウェアには限定されず、単一のハードウェア上で実現される複数のプロセスとして構成してもよい。また、エージェントは一つには限定されず、複数のエージェントが同時並行的に動作するようにしてもよい。また、情報処理の内容はファイルの操作には限定されず、計算、推論や他のノードとの通信など、自由に定めうる。

【0287】また、ローカル情報の内容は、ファイルが存在するかも知れないノード名には限定されず、構成要素にアクセスするためのドメイン名、ホスト名、ディレクトリ名、アクセス用ID、パスワードなど、自由に定める。

【0288】エージェント情報の内容やプランの表現形式も、前記実施形態に示したものには限定されず、自由に定めることができる。例えば、プランを表現する場合、因果リンクでアクションを接続する形式を取らず、アクションの系列をリスト構造やツリー構造で保存してもよい。また、第2実施形態において、失敗への対応の具体的手法も例示に過ぎず自由に定めることができる。また、本発明の実施において、知識ベースをフィールドに区分したり、エージェント固有の情報を用いたり、移動命令に事前条件を添付したりという構成は、必ずしも用いる必要はない。また、本発明の実施において、子エージェントの生成や、実行失敗時の再プランニングや、実行失敗時に原因となった情報を修正する構成は、必ずしも用いる必要はない。

【0289】また、適用される課題の性質に応じて、必ずプラン生成が成功して実行を要するような要求記述を入力する場合は、かならずしも判定手段を設ける必要はない。また、エージェントは必ずしもノード上のプロセスとして設定する必要はなく、専用のハードウェアやソフトウェアによって実現することもできる。

【0290】なお、本発明の情報処理装置及び情報処理方法は、典型的にはコンピュータプログラムを用いて実現されるが、そのようなプログラムを記録した記録媒体も本発明の一態様である（請求項7、14、15）。

【0291】

【発明の効果】以上説明したように、本発明によれば、ソフトウェアの構成要素の変化に柔軟に対応し、かつ、回線障害に対する耐障害性に優れた情報処理装置及び情報処理方法を提供することができる。

【図面の簡単な説明】

【図1】本発明の第1実施形態における情報処理装置の構成を示す機能ブロック図。

【図2】本発明の第1実施形態における情報処理の処理手順を示すフローチャート。

【図3】本発明の第1実施形態におけるプラン生成の手順を示すフローチャート。

【図4】本発明の第1実施形態において生成されたプランの内容を示すツリー図。

【図5】本発明の第1実施形態におけるエージェント転送の手順を示すフローチャート。

【図6】本発明の第1実施形態における表示例を示す図。

【図7】本発明の第2実施形態において、ノードの構成を示す機能ブロック図。

【図8】本発明の第2実施形態において、情報処理装置

に含まれるノードの構成を示すブロック図。

【図9】本発明の第2実施形態において、ノードがフィールドに区分されている状態を示す概念図。

【図10】本発明の第2実施形態において、エージェントの動作を構成する複数のフェーズを示す図。

【図11】本発明の第2実施形態において、エージェントの論理的な構造を示す図。

【図12】本発明の第2実施形態において、ゴール・スクリプト・セットのデータ構造を示す図。

【図13】本発明の第2実施形態において、ノードマネージャに係る通信を実現する態様を示す機能ブロック図。

【図14】本発明の第2実施形態において、知識ベースの構造を示す図。

【図15】本発明の第2実施形態において、ノードマネージャの動作手順を示すフローチャート。

【図16】本発明の第2実施形態において、プランニングの内容をフローチャート形式で示す図。

【図17】本発明の第2実施形態において、プラン実行の処理手順を示すフローチャート。

【図18】本発明の第2実施形態において、エージェントをノード間で移動させる場合に、移動元側のノードマネージャの動作手順を示すフローチャート。

【図19】本発明の第2実施形態において、エージェントをノード間で移動させる場合に、移動先側のノードマネージャの動作手順を示すフローチャート。

【図20】本発明の第2実施形態において、ノードマネージャを介したエージェントのノード間移動の状態を示す図。

【図21】本発明の第2実施形態において、エージェントの動作手順を示すフローチャート。

【図22】本発明の第2実施形態において、プラン実行への対処内容を示すフローチャート。

【図23】従来の情報処理装置の一例について、その構成を示す機能ブロック図。

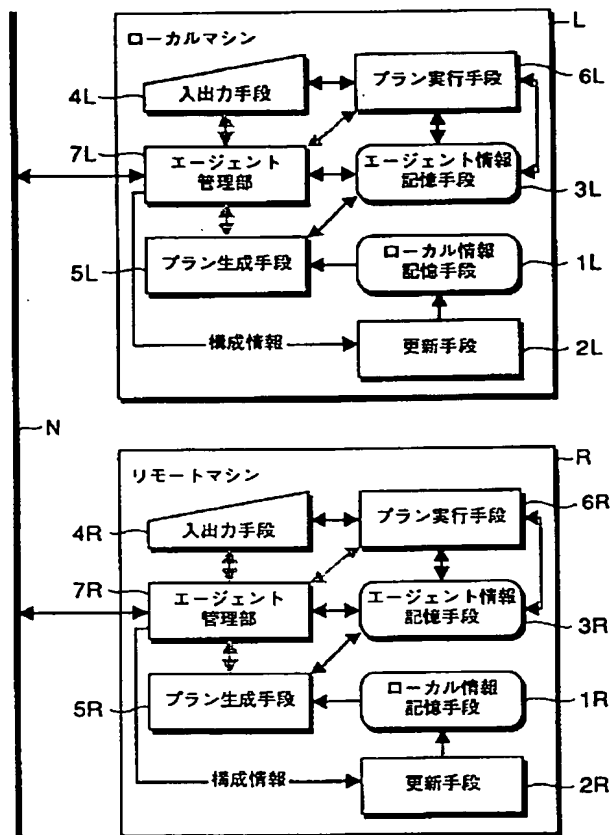
【符号の説明】

- 1…ローカル情報記憶手段
- 2…更新手段
- 3…エージェント情報記憶手段
- 4…入出力手段
- 5…プラン生成手段
- 6…プラン実行手段
- 7…エージェント管理部
- L…ローカルマシン
- R…リモートマシン
- N…ネットワーク
- STEP…手順の各ステップ
- 11、12、D…知識ベース
- 13…プランナ
- 14…実行器

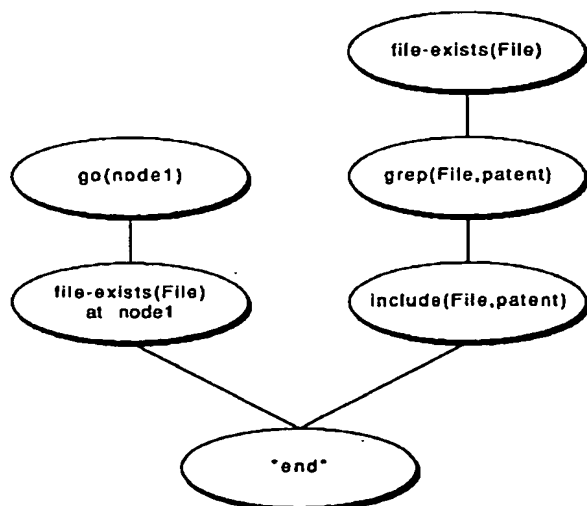
15…ノードマネージャ
 16…更新器
 17…エージェントプロセス
 18…GUIアプリケーション
 19…ノード管理情報記憶手段

X…ホスト
 FL…フィールド
 A…エージェント
 1501以降…手順の各ステップ

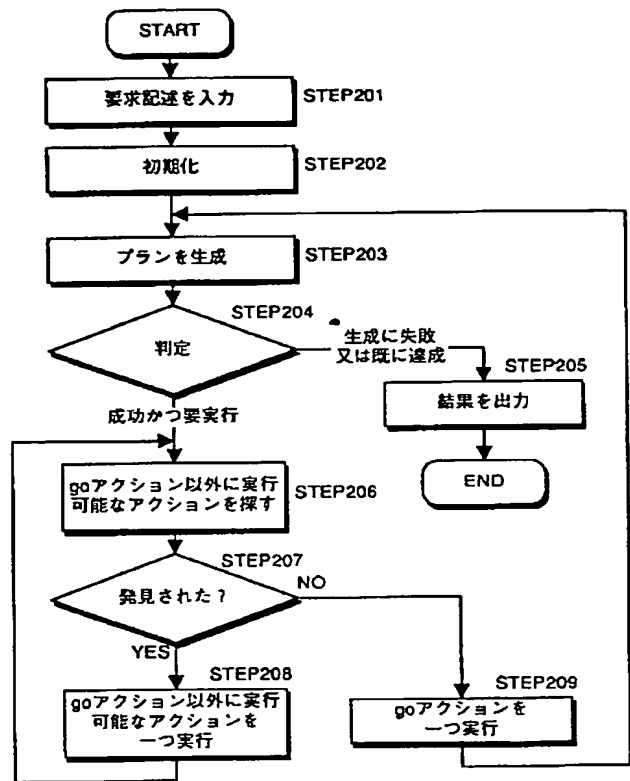
【図1】



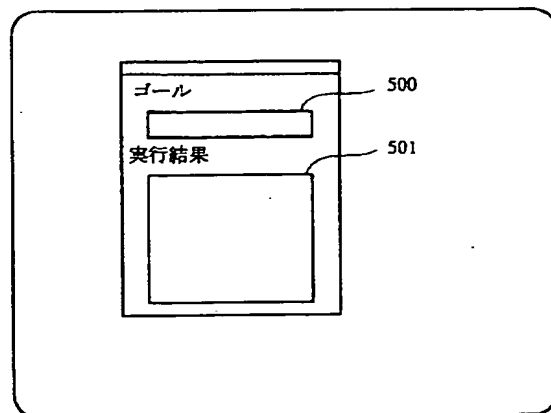
【図4】



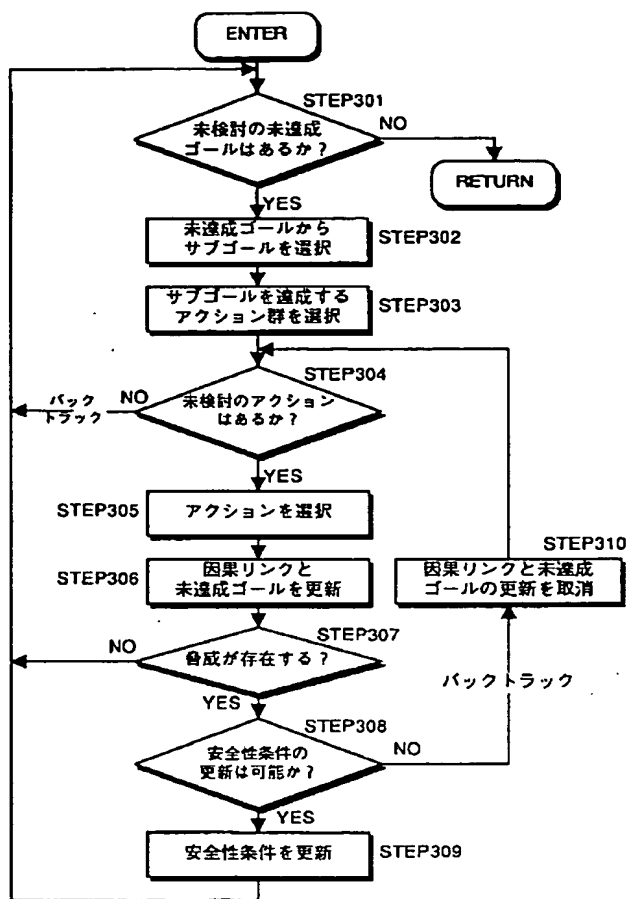
【図2】



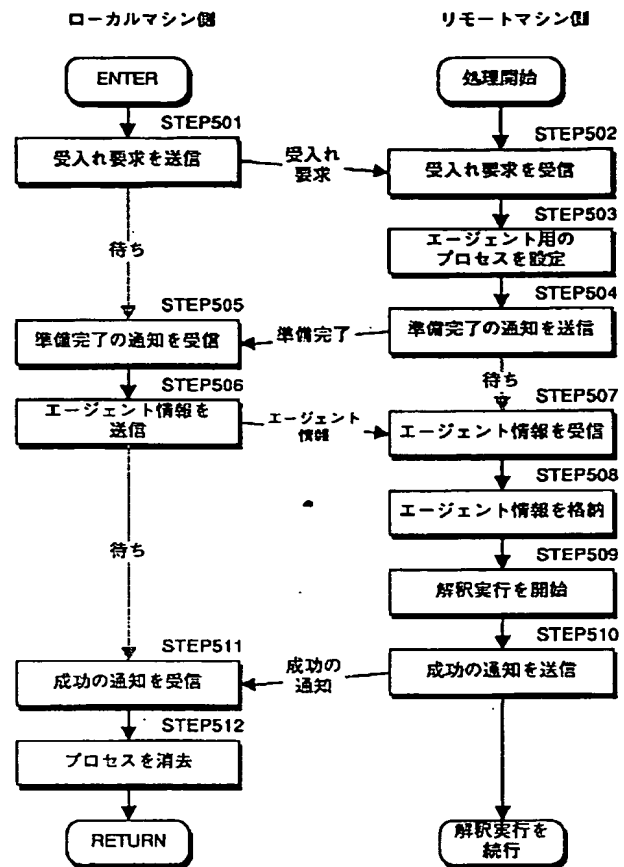
【図6】



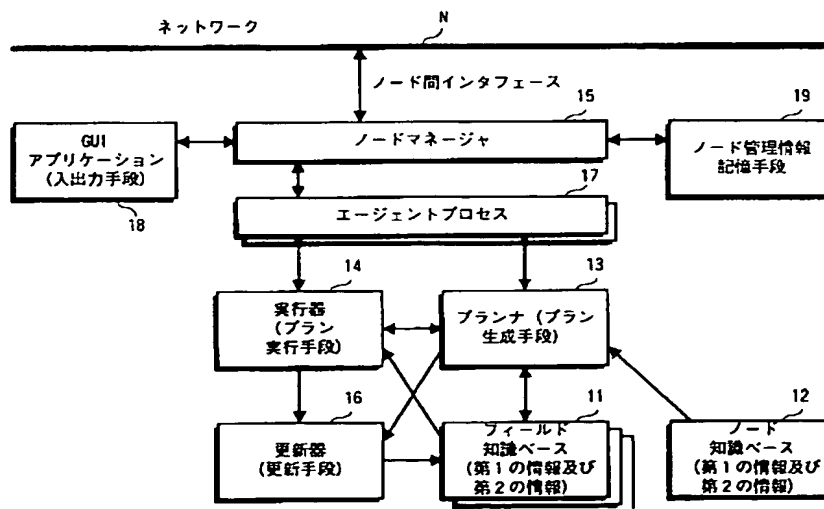
【図3】



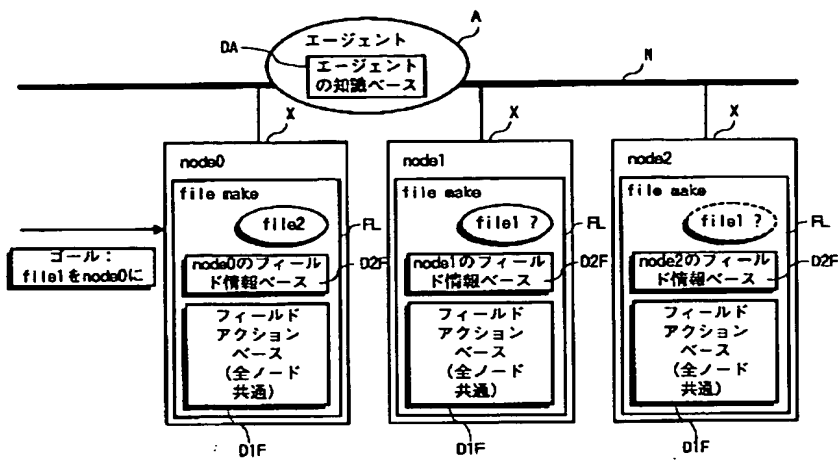
【図5】



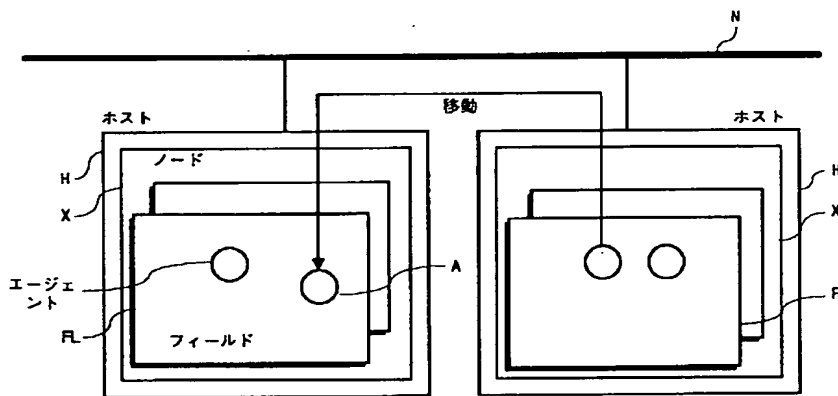
【図7】



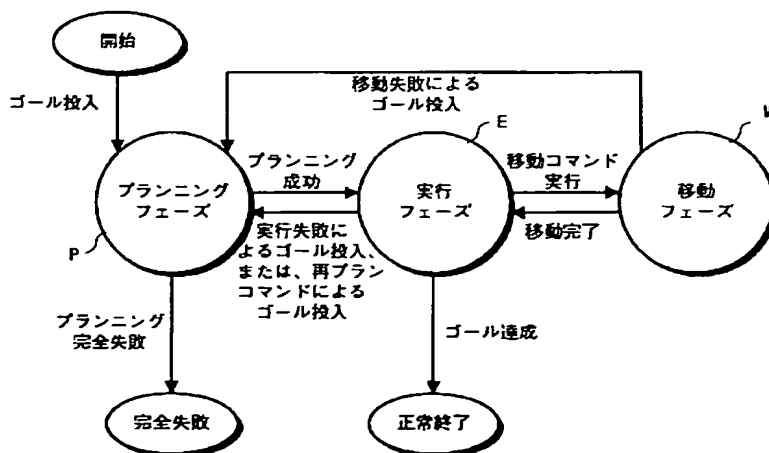
【図8】



【図9】



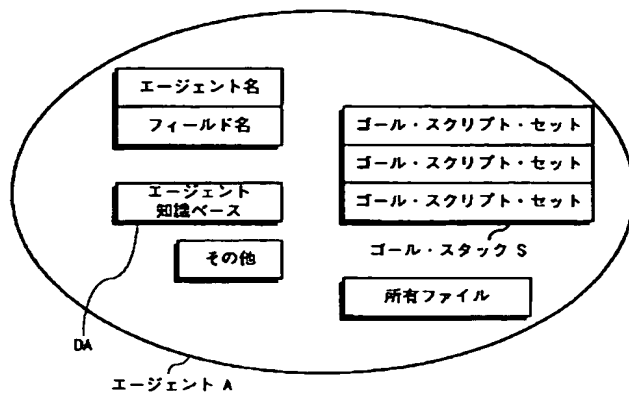
【図10】



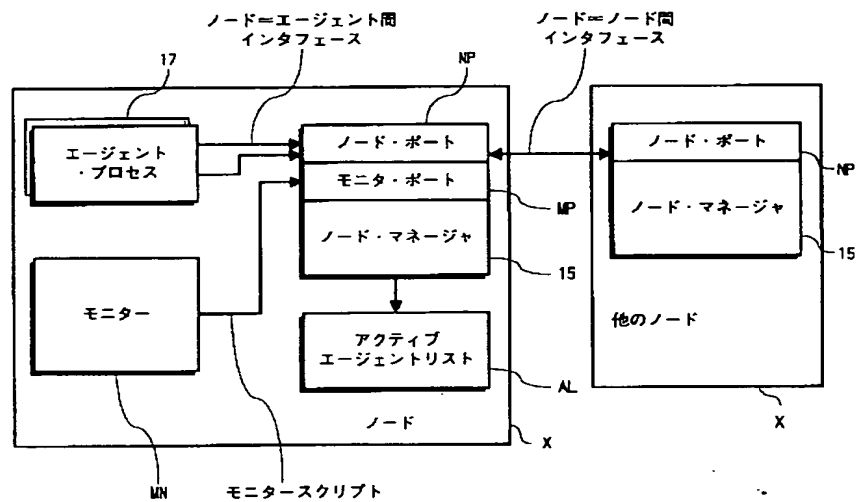
【図12】

ゴール	スクリプト	スクリプト変数値	スクリプト実行位置
-----	-------	----------	-----------

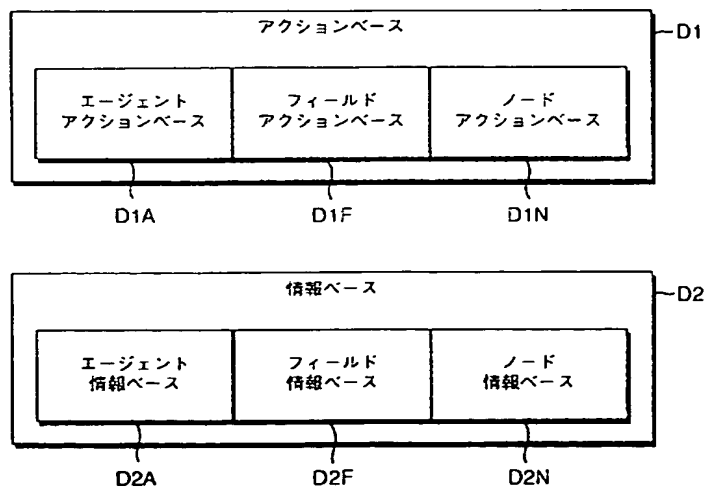
【図11】



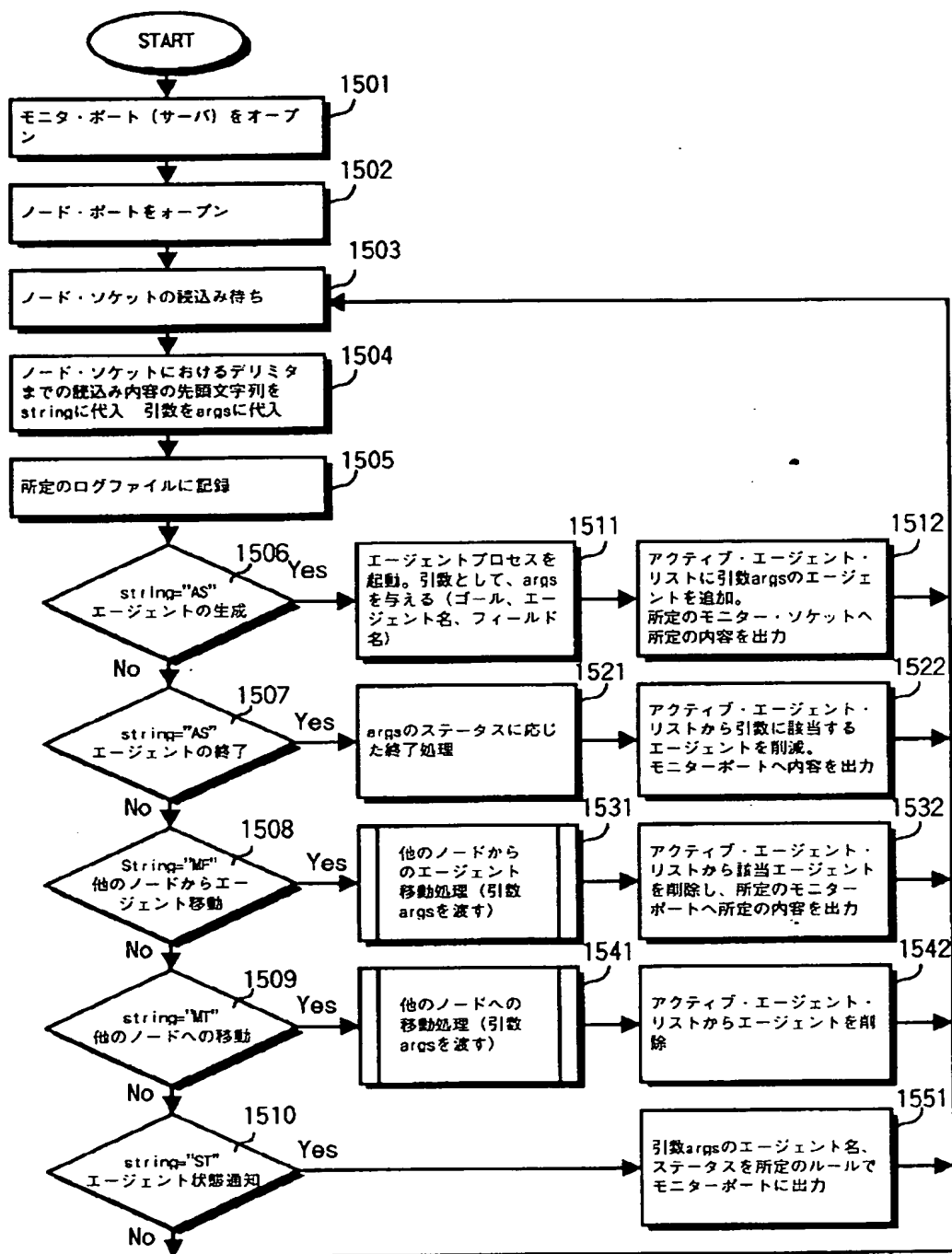
【図13】



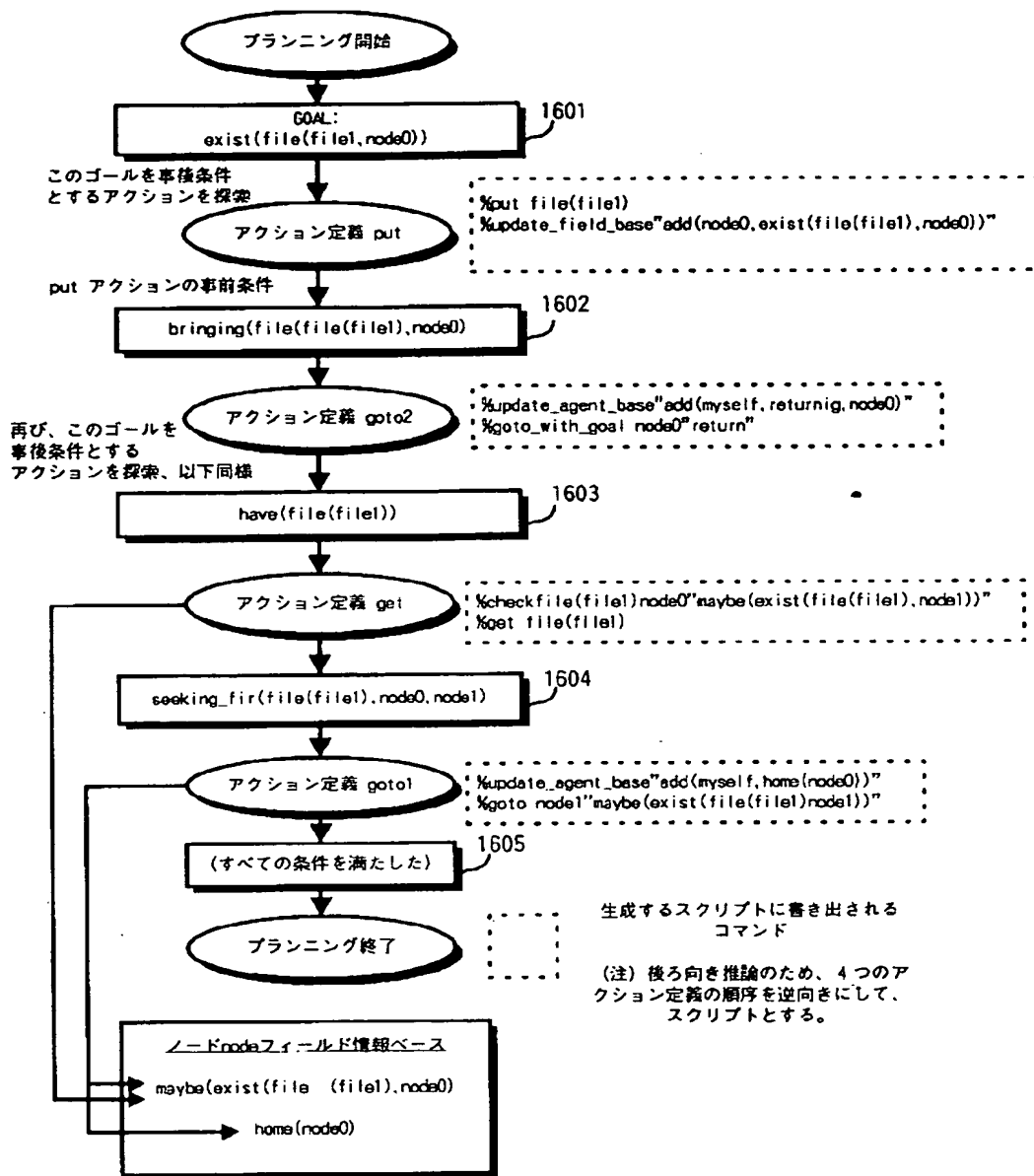
【図14】



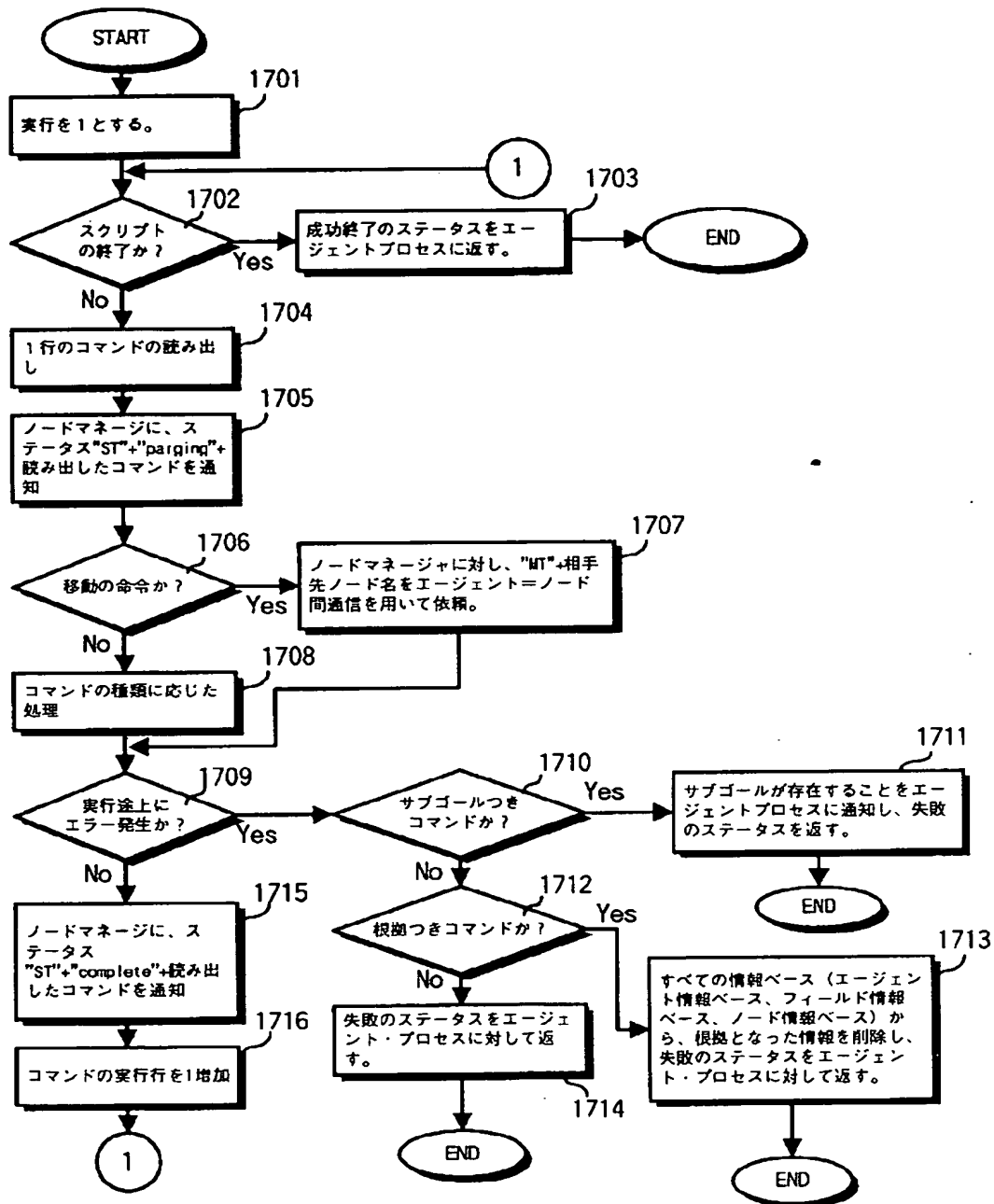
【図15】



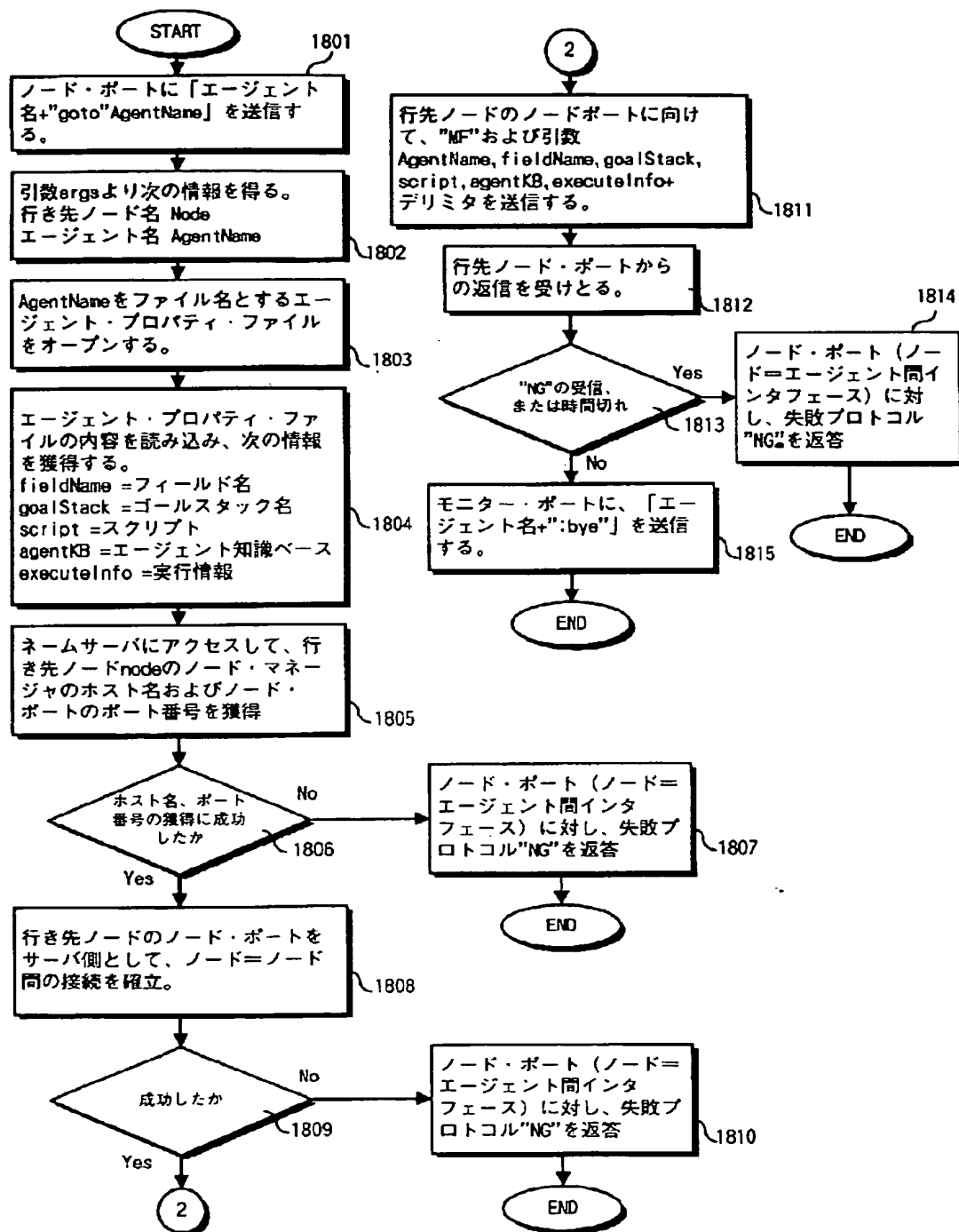
【図16】



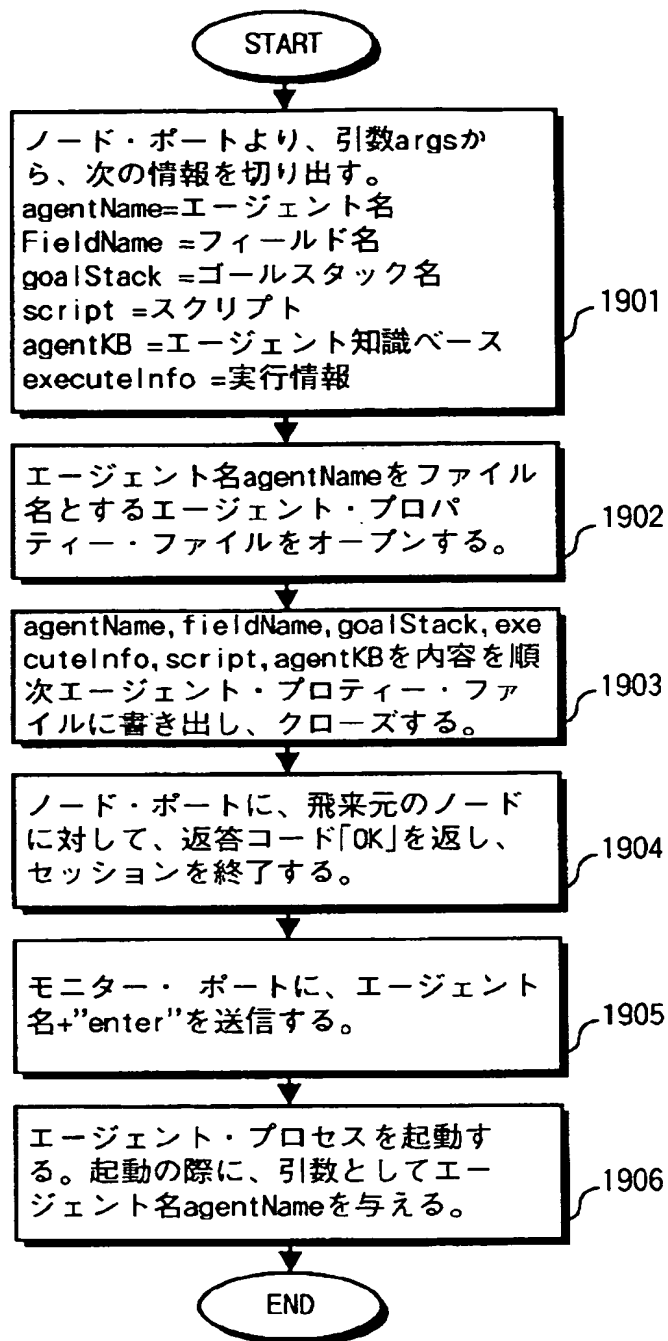
【図17】



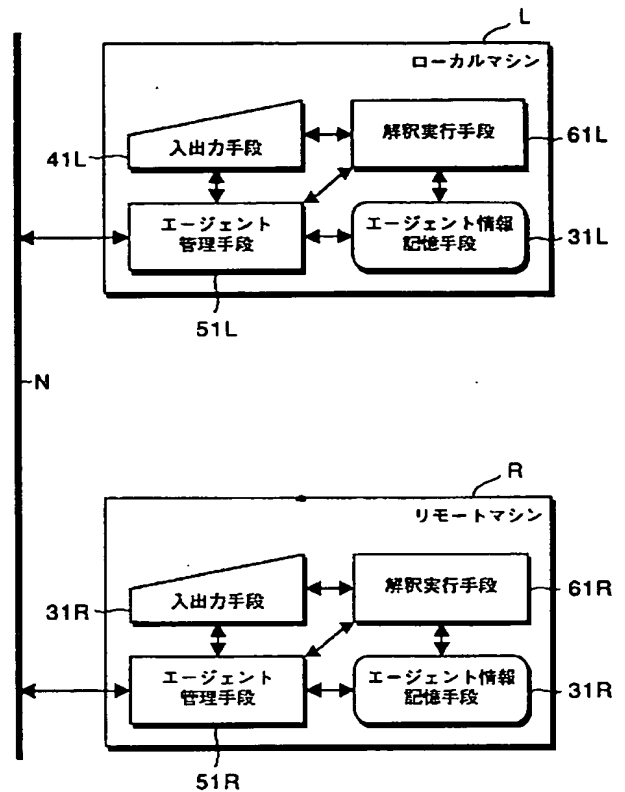
【図18】



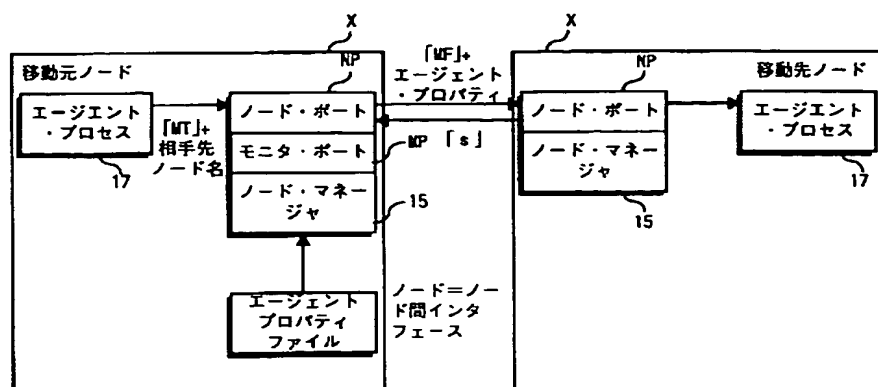
【図19】



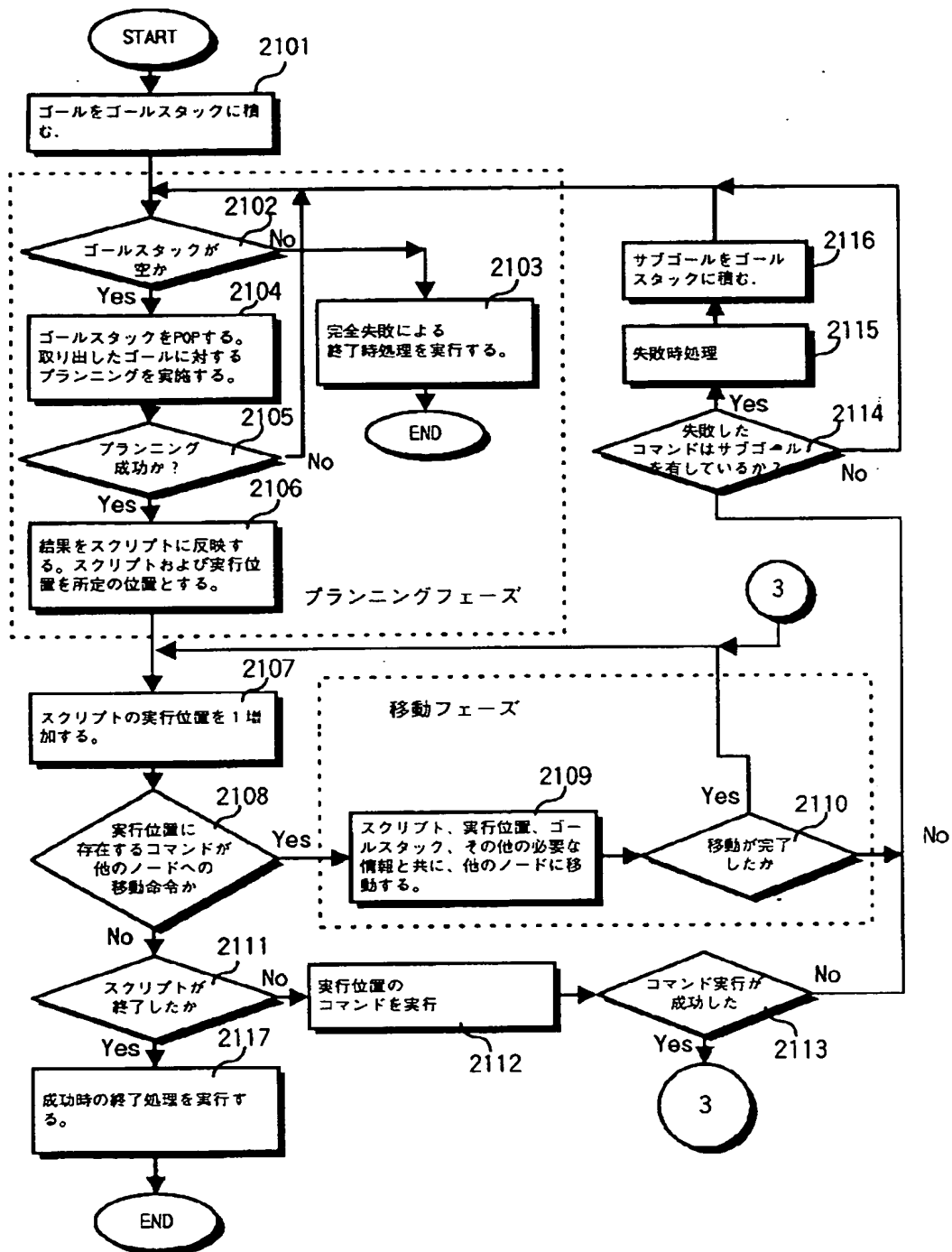
【図23】



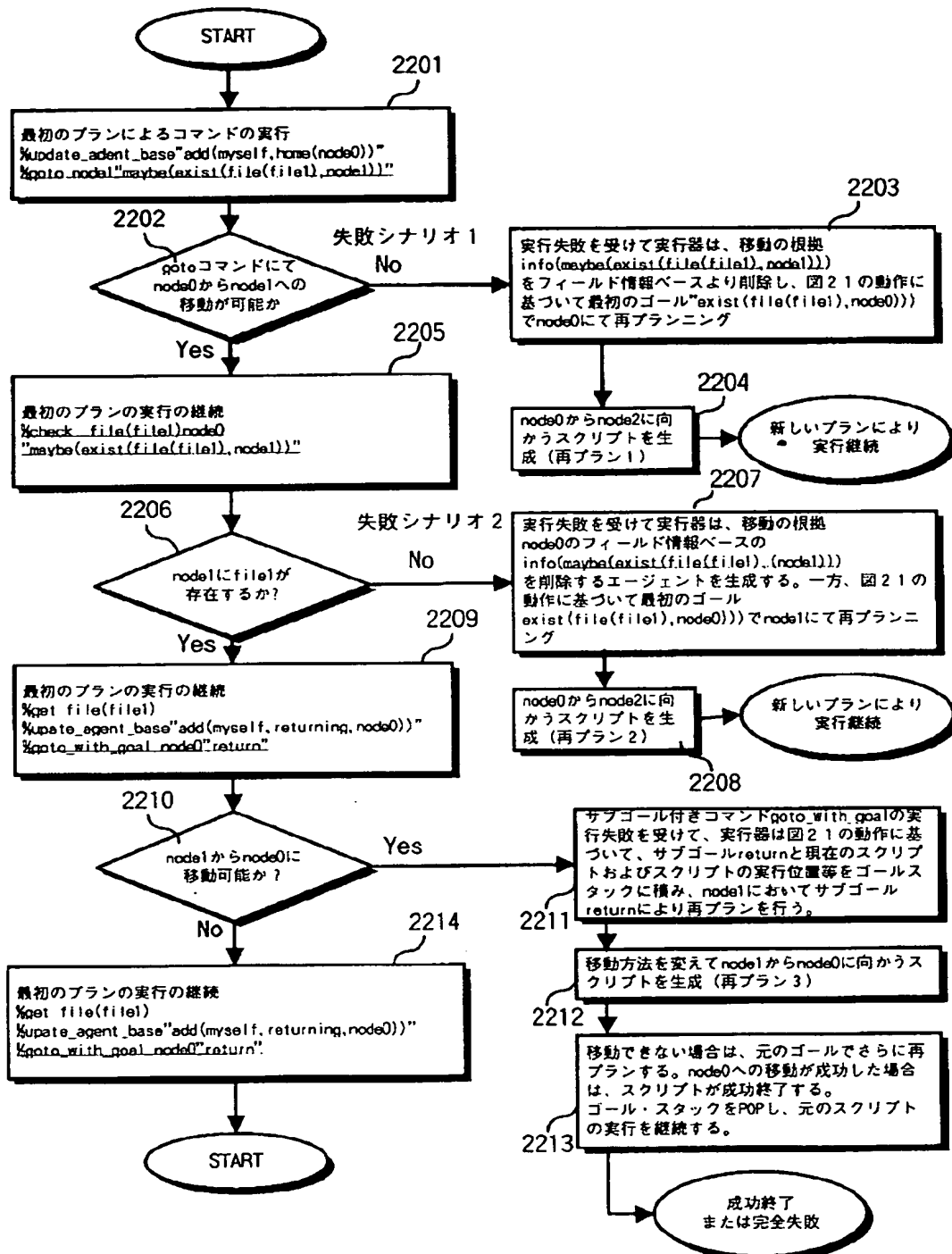
【図20】



【図21】



【図22】



フロントページの続き

(72) 発明者 加賀谷 聡
東京都府中市東芝町1番地 株式会社東芝
府中工場内

(72) 発明者 本位田 真一
神奈川県川崎市幸区柳町70番地 株式会社
東芝柳町工場内

(72) 発明者 入江 豊
神奈川県川崎市幸区柳町70番地 株式会社
東芝柳町工場内

(72) 発明者 服部 正典
神奈川県川崎市幸区柳町70番地 株式会社
東芝柳町工場内